НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем» спеціальності 121 Інженерія програмного забезпечення
на тему: «Децентралізована система для зберігання та поширення даних на основі технології S3»

Виконав:
студент IV курсу, групи КП-62
Кириченко Владислав Андрійович _____

Керівник:
Старший викладач кафедри ПЗКС
Бухтіяров Юрій Вікторович _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри ЕІ, к.т.н., доцент,
Вунтесмері Юрій Володимирович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

«ЗАТВЕРДЖУЮ»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___»_____2019 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Кириченко Владиславу Андрійовичу

1. Тема проєкту «Децентралізована система для зберігання та поширення даних на основі технології S3», керівник проєкту Бухтіяров Юрій Вікторович, затверджені наказом по університету від «25» травня 2020 р. № 1181-с

2. Термін подання студенткою проєкту «16» червня 2020 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз існуючих рішень;
   - обґрунтування вибору засобів реалізації;
   - опис розробленого веб-додатку;
   - аналіз розробленого додатку.

5. Перелік обов'язкового графічного матеріалу:
   - діаграма компонентів системи, архітектура системи (креслення);
   - схема бази даних системи (креслення);
   - компоненти та інтерфейси додатку (плакат);
   - дерево проблем (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2020 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	14.11.2019	
2.	Розроблення та узгодження технічного завдання	28.11.2019	
3.	Розроблення структури програмної платформи	15.12.2019	
4.	Підготовка матеріалів першого розділу дипломного проєкту	30.12.2019	
5.	Розроблення дизайну сторінок та графічних елементів	03.02.2020	
6.	Підготовка матеріалів другого розділу дипломного проєкту	20.02.2020	
7.	Програмна реалізація платформи	10.03.2020	
8.	Тестування програмної платформи	17.03.2020	
9.	Підготовка матеріалів третього розділу дипломного проєкту	30.03.2020	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	11.04.2020	
11.	Підготовка графічної частини дипломного проєкту	06.05.2020	
12.	Оформлення документації дипломного проєкту	01.06.2020	

Студент Владислав КИРИЧЕНКО

Керівник проєкту Юрій БУХТІЯРОВ

АНОТАЦІЯ

Даний дипломний проект присвячений розробленню децентралізованого сервісу для зберігання та поширення даних.

У роботі виконано порівняльний аналіз існуючих рішень для зберігання даних, проаналізовано методи зберігання на власному пристрої, та у “хмарному” сховищі, обґрунтовано вибір технологій та допоміжних бібліотек для реалізації даного сервісу. Розроблений сервіс надає користувачам можливість здійснювати зберігання своїх файлів, поширення їх між власними та довіреними пристроями. Збір даних можливо здійснити вручну, або за допомогою автоматизованих інструментів. Процес зберігання здійснюється автоматично, відносно попередньо встановлених налаштувань. В результаті дані зберігаються у відокремленому місці, яке підключається до додатку та є захищеним для від інших програм що запущені на пристрої.

В даному проекті розроблено та досліджено: архітектуру однорангового сервісу, алгоритм автоматичного та ручного збору інформації, сучасні методи зберігання даних у початковому вигляді, а також методи поширення цих даних.

ABSTRACT

This diploma project is dedicated to the development of a decentralized service for data storage and storage.

The paper performs a comparative analysis of existing solutions for data storage, analyzes storage methods on their own device and in the "cloud" storage, substantiates the choice of technologies and auxiliary libraries for the implementation of this service. The developed service gives users the opportunity to store their files, distribute them between their own and trusted devices. Data collection can be done manually or with automated tools. The storage process is performed automatically, relative to the preset settings. As a result, the data is stored in a separate place that connects to the application and is protected from other programs running on the device.

This project develops and researches: peer-to-peer service architecture, automatic and manual information collection algorithm, modern methods of data storage in its original form, as well as methods of data dissemination.

ДП.045440-01-90 Децентралізована система для зберігання та поширення даних на основі технології S3. Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Децентралізована система	5	
	для зберігання та		
	поширення даних на		
	основі технології S3.		
	Технічне завдання		
ДП.045440-03-81	Децентралізована система	50	
	для зберігання та		
	поширення даних на		
	основі технології S3.		
	Пояснювальна записка		
ДП.045440-04-51	Децентралізована система	4	
	для зберігання та		
	поширення даних на		
	основі технології S3.		
	Програма та методика		
	тестування		
ДП.045440-05-34	Децентралізована система	7	
	для зберігання та		
	поширення даних на		
	основі технології S3.		
	Керівництво користувача		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-06-99	Децентралізована система	1	
	для зберігання та		
	поширення даних на		
	основі технології S3.		
	Діаграма компонентів		
	системи. Архітектура		
	системи		
ДП.045440-07-99	Децентралізована система	1	
	для зберігання та		
	поширення даних на		
	основі технології S3.		
	База даних системи.		
	Схема бази даних		
ДП.045440-08-98	Децентралізована система	1	
	для зберігання та		
	поширення даних на		
	основі технології S3.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2019 р.

ДЕЦЕНТРАЛІЗОВАНА СИСТЕМА ДЛЯ ЗБЕРІГАННЯ ТА ПОШИРЕННЯ ДАНИХ НА ОСНОВІ ТЕХНОЛОГІЇ S3

Технічне завдання

ДП.045440-04-51

«ПОГОДЖЕНО»
Керівник проєкту:

_____ Юрій БУХТІЯРОВ

Нормоконтроль:
_____ Микола ОНАЙ

Виконавець:
_____ Владислав КИРИЧЕНКО

2019

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Децентралізована система для зберігання та поширення даних на основі технології S3.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проектування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання в якості власного децентралізованого та незалежного сховища даних. З метою здешевити та підвищити надійність зберігання даних, покращення показників швидкості доступу до даних.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Додаток повинен забезпечувати такі основні функції:

1. Забезпечити можливість користувачеві власноруч створити та запустити власне сховище, доступ до якого спочатку матиме лише він, але зможе надавати та налаштовувати доступ іншим.
2. Можливість об'єднати декілька окремих сховищ між собою для синхронізації вмісту при здійсненні операцій додавання та вилучення даних та підвищення надійності системи в цілому.
3. Підтримка актуальних на даний момент форматів даних.

4. Використання найбільш оптимізованих, швидких та надійних засобів для зберігання та обробки даних.

Розробку виконати на платформі Node.JS з використанням платформи Minio для зберігання даних.

5. ВИМОГИ ДО ПРОЕКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

1. Пояснювальна записка.
2. Програма та методика тестування.
3. Керівництво користувача.
4. Креслення:
   - «Діаграма компонентів системи. Архітектура системи»;
   - «База даних системи. Схема бази даних».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою роботи……..……..14.11.2019

Розроблення та узгодження технічного завдання……..28.11.2019

Розроблення структури додатку.....…….....……....15.12.2019

Розроблення архітектури додатку.....…….03.02.2020

Програмна реалізація додатку.....…….....……....17.03.2020

Тестування розробленого додатку...…….....…….....03.04.2020

Підготовка матеріалів текстової частини проєкту…….28.04.2020

Підготовка матеріалів графічної частини проєкту……..…12.05.2020

Оформлення технічної документації проєкту……...…25.05.2020

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до «Програми та методики тестування».

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

ДЕЦЕНТРАЛІЗОВАНА СИСТЕМА ДЛЯ ЗБЕРІГАННЯ ТА ПОШИРЕННЯ ДАНИХ НА ОСНОВІ ТЕХНОЛОГІЇ S3

Пояснювальна записка

ДП.045440-03-81

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Юрій БУХТІЯРОВ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Владислав КИРИЧЕНКО

2020

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення.

AWS (англ. Amazon Web Services) – набір онлайн сервісів від компанії Amazon для виконання різного роду задач використовуючи віддалені сервіси з високою швидкодією.

S3 (англ. Simple Storage Service) – засіб для зберігання документів у початковому вигляді та кодуванні [1].

Minio – проєкт з відкритим програмним кодом, розроблений як більш швидкий ніж AWS S3 сервіс конкретно для зберігання файлів.

Хмарне сховище – модель схову даних, де цифрові дані зберігаються в логічні пули, а фізичне зберігання охоплює кілька серверів (і часто на різних місцях).

Тунелювання – створення доступу до певних портів свого пристрою для запитів з глобальної мережі шляхом генерування унікального посилання, що вестиме на задані порти пристрою, дозволяє використовувати свій пристрій як хостинг для веб-сайтів.

SSH (англ. Secure Shell) – протокол, що дозволяє налаштувати віддалений доступ та керування іншим комп’ютером та передавати дані між ними [2].

p2p (англ. peer-to-peer, піринґова, однорангова мережа) – інформаційна система розроблена таким чином, щоб її функціонування не залежало від кількох центральних серверів, які проводять усі розрахунки та контроль процесу, а була рівномірно розподілена між усіма учасниками мережі, що значно підвищує надійність такої системи.

Checksum – певне значення, розраховане на основі вхідних даних за допомогою певної математичної хеш-функції, це забезпечує ідентичність результату виконання функції, при умові правильності вхідних даних.

NAT (англ. Network Address Translation) – механізм комп’ютерної мережі, а саме її елемента – маршрутизатора, який відповідає за

маскування IP-адрес комп’ютерів у локальній мережі, коли вони здійснюють запити до глобальної мережі.

TCP (англ. Transmission Control Protocol) – один з основних протоколів обміну даними між пристроями мережі, передача даних відбувається тільки коли з’єднання між двома кінцевими вузлами (той хто зробив запит і той кому він адресований) встановлено, також TCP протокол гарантує повну доставку повідомлення у початковому вигляді та отримання відповіді.

UDP (англ. User Datagram Protocol) – один з ключових сучасних протоколів передачі даних у мережі, але на відміну від TCP він лише здійснює передачу повідомлення, без необхідності встановлення з’єднання то без очікування відповіді, не гарантує доставку повідомлення.

ОС – операційна система, набір інструкцій за допомогою якого забезпечуються зручна взаємодія користувача з комп’ютером шляхом надання інтерфейсів для взаємодії.

Хост – комп’ютер на якому запущено ПЗ.

REST API – архітектурний стиль створення веб-додатків, що описує чітко визначені та стандартизовані типи запитів даних та відповідей на них.

Шардинг – реплікація даних у базі даних для підвищення надійності зберігання та можливості використовувати транзакції.

Кешування – це процес зберігання копій файлів у кеші чи тимчасовому місці зберігання, щоб швидше отримати доступ до них.

Фреймворк – це програмне середовище спеціального призначення, своєрідний каркас, який використовується для того, щоб істотно полегшити процес об'єднання певних компонентів при створенні програм.

Плагін – незалежний від основної програми модуль, що використовує її базові можливості та на їх основі доповнює основну програму, додаючи до неї нові можливості.

Серіалізація – процес перетворення структури даних, яку розуміє та використовує програма, у послідовність байтів з метою збереження даних або їх транспортування.

Токен – спосіб передати інформацію про користувача, необхідну для автентифікації, що закодована у вигляді строкового значення.

Кросплатформність – здатність ПЗ виконувати свої функції не залежно від операційної системи та програмного оточення, що його використовує.

Інтерпретатор – спеціальна програма, котра вміє зчитувати, аналізувати та перетворювати програмний код у послідовність команд для комп’ютера.

ВСТУП

Основна мета дипломного проєкту – розробка програмного забезпечення для зручного та надійного зберігання даних, використовуючи для цього лише власний пристрій або пристрої що використовують це ж ПЗ. Програмне забезпечення дозволяє зберігати будь-які дані у відокремленій пам’яті пристрою динамічного або фіксованого розміру, недоступної для інших програм та прямого доступу до її файлів, з можливістю шифрування вмісту. Особливістю є можливість поширювати дані безпосередньо зі свого пристрою, не використовуючи хмарні сховища.

Наразі люди користуються багатьма різними онлайн-сервісами для зберігання власних файлів, такі сервіси здебільшого представляють з себе універсальні рішення для цієї та багатьох інших проблем. Через зручність та популярність такого способу зберігання даних, власники цих сервісів постійно змушені нарощувати внутрішні об’єми сховищ та обчислювальні можливості через постійно зростаючу кількість запитів, такі умови змушують розробників використовувати різного роду засоби для оптимізації: використання черг на виконання, регіонального розподілення функціональних одиниць, тощо. У більшості випадків обирають такі сервіси через їх зручність у зберіганні безмежної кількості фото чи відео не у пам’яті свого пристрою, щоб в результаті мати доступ до них з будь якого свого пристрою просто увійшовши в акаунт.

Гарантувати доступ незалежно від часу і відстані, та унеможливити несанкціонованого доступу до файлів користувача сторонніми особами є одними з головних вимог до таких сервісів, але часто користувачі по причині своєї ж неуважності самі ж і надають доступ до своїх даних, а так як зберігаються вони віддалено і система зроблена так щоб забезпечити максимально швидкий доступ до них, то буде досить складно вчасно обмежити доступ до них перед тим як вони потраплять до рук сторонніх осіб. Також нікуди не зникне проблема, коли вам саме зараз потрібно

завантажити певний файл, а у цей момент на сховищі яке ви використовуєте закінчилось вільне місце, або воно не дозволяє вам завантажити необхідний файл.

Тому для частини користувачів було б досить зручно використати наприклад інший власний пристрій для зберігання даних, щоб таке сховище фізично було вашим і щоб до даних мали б доступ лише ви, але водночас могли використовувати ті самі переваги що надають хмарні сховища, такі як доступ будь-коли та з будь-якої точки світу, щоправда цей підхід передбачає необхідність завчасно просто увімкнути власне сховище.

Програмне забезпечення розроблене в цьому дипломному проєкті надає такий набір функціональних можливостей:

- забезпечити можливість користувачеві власноруч створити та запустити власне сховище, доступ до якого спочатку матиме лише він, але зможе надавати доступ кому забажає;
- забезпечити декільком користувачам одночасно запустити декілька екземплярів ПЗ та будуть між собою автоматично синхронізуватись задля зменшення ризику пошкодження даних, а також одночасно завдяки шифруванню щоб у кожного користувача був власний простір;
- забезпечити можливість прямого з’єднання між учасниками без необхідності у наявності певного сервера, що буде їх між собою пов’язувати;
- реалізація власне функціоналу сховища у вигляді спеціальної програми, яка запускається на пристрої, виділяє для своїх потреб необхідну кількість пам’яті на пристрої та використовує її як єдиний блок, всередині він може мати безліч різноманітних даних, але для інших програм та при просто пошуку серед файлів усе сховище включно зі змістом представляє собою єдиний файл, який при потребі можна максимально просто зарезервувати, або при потребі видалити;

- підтримка актуальних на даний момент форматів даних;
- використання найбільш оптимізованих, швидких та надійних засобів для зберігання та обробки даних;
- реалізація можливості віддаленого доступу до даних через посилання на запущений на пристрої сервіс-сховище, можливості поширення та налаштування доступу до даних іншим користувачам;
- створення власне системи для використання описаних вище засобів через узагальнений API інтерфейс.

Разом вище перелічені можливості забезпечать максимально швидку та безпечну систему для зберігання та обробки даних. Не зважаючи на те, що даний тип програми не є єдиним рішенням на ринку ПЗ, в цьому дипломному проєкті буде проаналізовано вже існуючі рішення і розроблено останню версію продукту з урахуванням всіх мінусів існуючих розробок.

Дане ПЗ не потребує від користувача спеціальних технічних знань та навичок програмування, що робить його доступним для кожного.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1. Загальний опис проблеми зберігання даних

Сховище даних – набір даних, що зберігає послідовність зміни даних і здатний з максимальною швидкістю та ефективністю виконувати операції зчитування-запису та забезпечує зберігання інформації за рахунок комп’ютерів та інших пристроїв [3]. Найпоширенішими форматами зберігання даних є зберігання файлів, блокове зберігання та зберігання об’єктів, кожен з яких ідеально підходить для різних цілей.

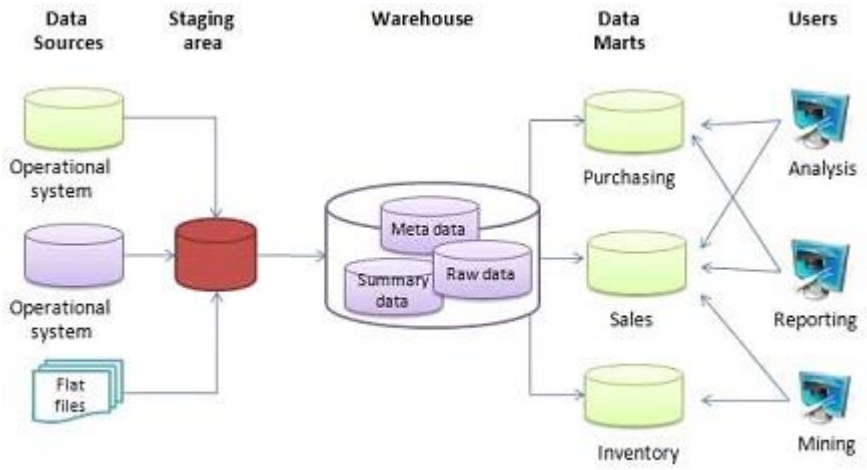


Рис. 1. Загальна структура сховищ даних

Проблеми створення сховищ даних:

- Дані потребують пам’яті для зберігання. Якщо потрібно зберігати велику кількість даних, з’являється необхідність у інфраструктурі для зберігання, що часто означає вкладення коштів. Одним із найпростіших способів вирішення проблеми є використання хмарних сховищ.
- Безпека є одним з головних питань, яке потрібно вирішити. Гіпотетично, якщо ваші дані десь зберігаються, їх може отримати третя сторона. Існує безліч шарів безпеки, які можуть допомогти запобігти цьому несанкціонованому доступу,

включаючи шифрування та надійність сторонніх постачальників, але існує обмеження на те, наскільки добре вони можуть захистити дані.

- Дані, що зберігаються можуть пошкодитись, такі явища як магнітні та електромагнітні поля здатні зіпсувати дані на пристрої, та зробити неможливим надалі їхнє використання. Навіть якщо немає зовнішнього джерела, яке безпосередньо йому заважає, дані можуть з часом погіршуватися. Найефективнішим вирішенням цієї проблеми є використання кількох резервних копій.
- Якщо потрібно використовувати для своїх даних кілька систем або додатків, потрібно переконатися, що вони сумісні. Для цього потрібно знайти сховище даних з відкритим API та підтримкою кросплатформності.

1.2. Аналіз існуючих рішень для даної задачі

Проаналізувавши готові рішення для поставленої задачі, було знайдено декілька прикладів, серед яких є досить відомі: Napster (1999 – 2001), протокол та однойменний клієнт для роботи з мережею – BitTorrent (2001 – по нині), Kademlia (2002 – по нині). Усі наведені у приклад компанії знайшли власні вирішення головної проблеми у створенні подібного роду сховищ – пошуку інших учасників мережі для встановлення зв’язку та обміну даними.

1.2.1. Napster

Napster – піринговий файлообмінник розроблений у США у 1999 році двома студентами Північно-Східного Бостонського університету, за рахунок появи на початку 21 століття одразу ж привернув до себе увагу користувачів, яким відкрилася можливість за допомогою нього обмінювались будь-якою музикою з іншими користувачами, що досить швидко призвело до закриття сервісу за рішенням суду, але стало

поштовхом для створення багатьох схожих сервісів, частина з них зараз стали потоковими медіа-сервісами.

Переваги:

- обмін файлами відбувався у безпосередньому з'єднанні між користувачами;
- вузькоспеціалізований сервіс для обміну та продажу музики у форматі MP3, враховуючи децентралізоване зберігання даних, та особливості транспортування даних між користувачами підтримував усі популярні платформи на свій час.

Недоліки:

- для налаштування роботи системи, а саме запит на список та адреси підключених користувачів, а також розповсюджуваних ними файлів використовувався центральний сервер і під час підключення нового користувача або ж пошуковому запиті сервер просто видавав адресу іншого користувача і далі вже сам обмін інформацією проходив безпосередньо, але враховуючи це складно відносити даний сервіс до насправді децентралізованих.

1.2.2. BitTorrent

BitTorrent – мережевий прокол, програмний код якого знаходиться у відритому доступі, призначений для обміну даними у peer-to-peer мережах, повноцінно реалізує поняття децентралізованої мережі, проте має свої певні особливості. При роботі мережі в ній приймають участь наступні основні групи учасників:

- трекер (англ. tracker) – спеціалізований сервер, що здійснює координацію запитів користувачів мережі BitTorrent, зберігає адреси користувачів та фрагменти даних, які вони раніше звантажили і зможуть підтримувати роздачу;
- піри (англ. peer) – учасники мережі, які завантажують потрібні їм дані та одночасно відкриваюсь доступ іншим користувачам до інших фрагментів;

- сіди (англ. seeder) – такий користувач (peer) що вже повністю завантажив собі потрібний файл і здатний відтворити роздачу цього файлу іншим, навіть якщо більше немає користувачів, що діляться ним;
- роздача (seeding) – власне процес при якому користувач отримує необхідні йому дані та одночасно з цим починає розповсюджувати їх іншим учасникам мережі для забезпечення максимальної доступності.

Переваги:

- Обмін даними файлів або просто метаданих, необхідних усій системі для функціонування відбувається постійно за замовчуванням, якщо спеціально вимкнути цю опцію. Такий підхід дозволяє різним користувачам у яких є певний файл або ж вони його лише завантажують у даний момент, вже ділитись даними з іншими учасниками мережі. Це реалізується завдяки поняттю контрольних сум (checksum), яке вираховується для кожного фрагменту даних та забезпечує отримання усіх необхідних даних при передачі їх фрагментами.
- У мережі окрім самих учасників також існують трекери-вузли (спеціальні сервери або інші користувачі), які займаються зберігання адрес учасників мережі та підрахунком контрольних сум, проте у сучасних версіях користувачі майже повністю автономні.

Недоліки:

- Мережа що працює за таким принципом де обмін даними фактично буде продовжуватись допоки до неї залишається приєднаним хоча б один сід, у випадку якщо дані просто ніхто не роздає то і втрачається можливість надалі їх отримати, лише якщо на роздачу повернеться сід.

- Наявність та необхідність у мережі трекерів іноді призводить до ситуацій коли трекер з певних причин недоступний і нові користувачі просто не зможуть приєднатися до роздачі.

1.2.3. Kademlia

Kademlia – реалізація альтернативного підходу до створення децентралізованої мережі, мережа існує за рахунок хеш-таблиці, яка розподілена одразу між усіма учасниками мережі та відповідає за усі адреси всередині неї, тобто при звантажені файлу він отримає унікальний ідентифікатор і надалі пошук шляху до нього перетвориться у пошук всередині хеш-таблиці за певним алгоритмом.

Переваги:

- при додаванні нового учасника до мережі, йому присвоюється унікальний ідентифікатор, потім відбувається пошук по мережі, з метою пошуку вузлів зі схожим ідентифікатором (розмір ідентифікатора файлу повинен співпадати з розміром ідентифікатора вузла), що забезпечить оптимізацію пошуку файлу при запиті;
- назви файлів поділено на складові частини до яких застосовано один і той самий алгоритм хеш-функції, що дозволяє досягти максимальної ефективності пошуку по ключовим словам.

Недоліки:

- розподілена хеш-таблиця потребує одночасних підрахунків з боку усіх учасників мережі, та може бути досить складною задачею.

1.3. Постановка задачі

Загальні вимоги до ПЗ розподіленого сховища даних:

- Можливість незалежно від ОС та ПЗ встановленого на пристрої завантажити та встановити необхідні для роботи сервіси, такі як S3 сервер Minio, який власне і забезпечує функціонування

сервісу, без встановлення його як повноцінної програми на комп’ютері, а лише як допоміжної.

- Підтримувати графічний або консольний інтерфейс для взаємодії з користувачем, перегляду вмісту сховища, створення тимчасових посилань на дані та інтерфейс для налаштувань комунікації зі сховищем.
- При запуску програми з необхідними налаштуваннями також створювати та завантажувати екземпляр серверу Minio у оперативну пам’ять, та при зупинці коректно його вимикати, щоб уникнути випадкового створення безконтрольного процесу в операційній системі.
- Надати можливість при бажанні підключитись до будь-якого іншого сховища, що використовує S3-подібну систему при потребі [4].
- Зберігати саме сховище і весь його зміст у вигляді файлу за замовчуванням, або у вигляді контейнеру при можливості запуску у такому режимі для зручного та безпечного зберігання даних.
- Реалізація функціоналу використовуючи потокову обробку даних з метою оптимізації використання ресурсів машини на якій сервіс буде запущено, покращення загальної швидкодії, підтримки декількох одночасно запущених операцій.
- Можливість налаштувати вмикання сховища на час роботи клієнту постійно очікуючи на підключення запуск при певних умовах та в певний час.
- Можливість постійної роботи у режимі коли потрібно надати доступ до певних портів свого пристрою (тих, на яких і запущено сервіс) з зовнішньої мережі через захищений канал який буде перенаправляти усі запити саме за вказаний екземпляр сервісу.

- Можливість ділитись даними з іншими людьми генеруючи спеціальні посилання та встановлюючи на них необхідні обмеження, так щоб можна було поділитись лише на деякий час або ж встановити певний ліміт на кількість завантажень, щоб при досягненні цього числа переглядів дані ставали недоступними.
- Можливість перевірити правильність даних, наприклад чи не є вони «биті» – такі, що під час їх збереження процес не був коректним чином завершеним і тому файл хоч і відповідає за своїми метаданими вказаному формату даних, проте насправді його вміст неможливо або складно буде прочитати використовуючи інструкцію для вказаного формату даних. Користувач повинен мати змогу одразу дізнатися що файл, який він хоче зберегти не відповідає вказаному формату і потім вирішити що з ним робити, це дозволить економити ресурси сервісу.
- Можливість оптимізації розміру даних, що зберігаються, шляхом попереднього стиснення їх та подальшої відтворення їх при запиті у потоковому режимі.
- Можливість перегляду та редагування метаданих, які зберігаються разом з файлами [5].

2. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Вибір мови програмування

2.1.1. JavaScript

JavaScript є однією з найпопулярніших мов програмування, що використовується в HTML та веб-розробках, одна з об'єктно-орієнтованих скриптових мов, яка також є легкою з точки зору пам’яті та підтримки для кросплатформної розробки. Він переважно підходить для створення динамічних сторінок на стороні клієнта для веб-сайту.

Будь-яка програма, написана на JavaScript називається скриптом. Ці сценарії використовуються на сторінках HTML, які будуть виконані автоматично під час завантаження сторінки. І ще одна річ про ці сценарії надається та виконується як звичайний текст і не потребує спеціальної підготовки чи компіляції для запуску. Програміст Netscape Communications Corporation Брендан Ейх розробив JavaScript та запустив його у вересні 1995 року, але спочатку називав Mocha. Після здобуття репутації найкращого інструменту для описання скриптів він був перейменований на JavaScript, що відображає підтримку Netscape Java в її браузері.

Особливості мови JavaScript:

- зв'язок із сервером, дозволяє перевірити вхід користувача перед тим, як сторінка буде відправлена на сервер;
- наявність величезної кількості API, що дозволяють розробникам інтегрувати у власні сайти функціонал інших провайдерів, таких як Twitter або Facebook;
- фреймворки та бібліотеки, які доступні для застосування покликані прискорити створення сайтів та застосунків.

2.1.2. TypeScript

TypeScript є чистою об'єктно-орієнтованою мовою програмування з відкритим кодом. Це одне з найбільш вагомих розширень можливостей мови JavaScript, також програми на TypeScript можна без особливих

проблем перетворити на звичайний JavaScript. TypeScript не працює безпосередньо в браузері. Перш ніж його потрібно скласти для компіляції та генерації у файлі JavaScript. Вихідний файл TypeScript знаходиться у розширенні ".ts", яке дозволяє запускати на будь-яких пристроях. Будь-хто може використовувати файл .js для перевірки файлу TypeScript, перейменувавши його у файл .ts.

Microsoft за ліцензією Apache 2. розробив TypeScript, а також підтримував його. Андерс Хейльсберг розробив TypeScript. Вперше він був представлений громадськості в місяці 1 жовтня 2012 року. Microsoft випустила нову версію TypeScript 0.9 у 2013 році після двох років внутрішньої розробки. Поточна версія TypeScript 3.4.5, яка вийшла 24 квітня 2019 року.

Переваги над JavaScript:

- можливість використовувати статичну типізацію під час створення ПЗ;
- підтримка абстракцій, інкапсуляції, наслідування, та поліморфізму, на рівні з цілком об’єктно-орієнтованими мовами програмування.

За задумом ці нововведення мають підвищити швидкість розробки, зрозумілість, покращення і повторне використання коду, здійснювати пошук помилок на етапі розробки та компіляції, а також швидкодію програм. TypeScript, а також JavaScript побудований з власними обмеженнями та можливостями. JS – це легка та динамічна мова кодування, особливо для вдосконалення веб-сторінок HTML. Однак це не повноцінна мова кодування. Як інтерпретована мова програмування, JS потрапляє в контекст веб-браузера. Більш того, можна залишити існуючі JavaScript-проєкти в незмінному вигляді, а дані про типізації розмістити у вигляді анотацій, які можна помістити в окремі файли, які не заважатимуть розробці і прямому використанню проєкту (наприклад, подібний підхід зручний при розробці JavaScript-бібліотек).

2.1.3. Python

Python є популярною та широко використовуваною мовою програмування, велика кількість проєктів GitHub, використовують Python. Інтерпретатор мови Python працює на всіх основних операційних системах і платформах, і на більшості другорядних. У багатьох великих бібліотеках та службах, що працюють на основі API, мають прив'язки або обгортки Python, що дозволяє Python вільно взаємодіяти з цими службами та безпосередньо використовувати ці бібліотеки.

Основний напрямок використання Python – для опису різноманітних сценаріїв та автоматизації. Важливим аспектом Python є його динамізм. Все в мові, включаючи самі функції та модулі, обробляються як об'єкти. Це дозволяє набагато простіше писати програмний код високого рівня. Розробники можуть виконувати складні маніпуляції з об'єктом лише за допомогою декількох інструкцій і навіть трактувати частини програми як абстракції, які за потреби можуть бути змінені.

Переваги над JavaScript та TypeScript:

- простий у написанні програмний код, в якому розробнику не потрібно тримати на увазі розподіл пам’яті, поведінку програмних компонентів під час виконання, тощо;
- підтримка великої кількості різноманітних бібліотек та фреймворків.

Недоліки:

- відсутність асинхронної моделі обробки даних у мові програмування за замовчуванням;
- необхідність використовувати різні мови програмування при написанні графічного інтерфейсу для користувача та власне самого застосунку.

2.1.4. Висновки

Таблиця 1

Порівняння мов програмування JavaScript, TypeScript, Python

Назва	JavaScript	TypeScript	Python
Необхідність компіляції	Ні	Так	Ні
Статична типізація	Ні	На етапі компіляції	Так
Можливість написання серверного коду	Так	Так	Так
Асинхронна обробка даних	Так	Так	Ні

Таким чином основною мовою програмування для розроблюваного ПЗ обрано TypeScript, як оптимальний варіант між наявністю статичних типів як у Python та високоефективною асинхронною моделлю виконання операцій, як у JavaScript. Також TypeScript повністю підтримується у Node.JS що робить її універсальним вирішенням практично усіх проблем підчас розроблення програмного продукту.

2.2. Вибір фреймворку для розробки серверної сторони додатку

2.2.1. Node.js

Node.JS – проєкт з відкритим кодом, що використовує, кросплатформене середовище виконання JavaScript. Він виконує код JavaScript за межами браузера. Проєкт Node.js використовує модель відкритого управління. Фонд OpenJS забезпечує підтримку проєкту.

Особливості:

- асинхронна обробки операцій вводу-виведення даних;

- система модулів що підтримує та значно розширює систему модулів у JavaScript та TypeScript;
- використовує кросплатформений засіб для запуску JavaScript скриптів – Google V8.

2.2.2. Electron

Electron – фреймворк, для побудови кросплатформених програм, які встановлюються та працюють безпосередньо на комп'ютерів, використовує платформу NodeJS для запуску, та додає до неї багато нових можливостей. Electron – це основа для створення настільних додатків з усіма новими технологіями, включаючи JavaScript, HTML та CSS. В основному, система Electron піклується про важкі деталі, щоб ви могли зосередитись на суті програми та зробити революцію в її дизайні. Розроблений як рамка з відкритим кодом, Electron поєднує в собі найкращі веб-технології та є кросплатформеною, – це означає, що вона легко сумісна з Mac, Windows та Linux. Він постачається з автоматичними оновленнями, власними меню та сповіщеннями, а також повідомленнями про аварійне завершення роботи, налагодженням та профілюванням.

Особливості:

- розширює стандартну бібліотеку Node.JS функціями, що відповідають за різноманітні події під час взаємодії користувача зі звичайними програмами на комп'ютері.

2.2.3. Висновки

Таблиця 2

Порівняння фреймворків для розробки серверної частини Node.JS, Electron

Назва	Node.JS	Electron
Створення програм, що підтримують взаємодію з користувачем	Так	Так

Продовження табл. 2

Взаємодія з користувачем через віконний інтерфейс	Ні	Так
Необхідність у сторонніх засобах для роботи	Ні	Так
Можливість викликати сторонні програми	Так	Так

Таким чином основним фреймворком для розробки серверної частини додатку обрано звичайний Node.JS, тому що Electron, як доповнення для Node.JS, що підтримує роботу настільних програм у операційній системі, потребує значно більше ресурсів для роботи, а наявність саме віконного інтерфейсу не є необхідністю. Для написання безпечнішого коду Node.JS буде використовуватись разом з TypeScript.

2.3. Вибір системи керування базами даних

2.3.1. PostgreSQL

PostgreSQL – це об'єктно-реляційна система управління базами даних (ORDBMS) з акцентом на розширюваність та відповідність стандартам. Як сервер бази даних, його основна функція – зберігати дані, надійно та підтримувати кращі практики, а також отримувати їх пізніше, як цього вимагають інші програмні програми, будь то ті, що працюють на тому ж комп'ютері, або ті, що працюють на іншому комп'ютері в мережі. Він може працювати з навантаженнями, починаючи від невеликих локальних програм до великих Інтернет-додатків з багатьма одночасно користувачами. Останні версії також забезпечують реплікацію самої бази даних для безпеки та масштабованості.

PostgreSQL реалізує більшість стандартів SQL: 2011, сумісний із ACID та трансакційним (включаючи більшість заяв DDL), уникаючи

проблем із блокуванням, використовуючи багатоверсійний контроль сумісності (MVCC), використовуючи безліч методів індексації, недоступних в інших базах даних; має оновлені види та матеріалізовані подання, тригери, зовнішні ключі; підтримує функції та збережені процедури та інші можливості розширення та має велику кількість розширень, написаних третіми сторонами.

Має наступні особливості:

- користувацькі типи даних;
- наслідування таблиць структур даних;
- обробка подій всередині бази даних;
- точки збереження всередині транзакцій;
- багатоверсійний контроль сумісності (MVCC);
- асинхронна реплікація.

2.3.2. MongoDB

MongoDB – це база даних документів, що означає, що вона зберігає дані в JSON-подібних документах. База даних підтримує велику кількість різноманітних запитів, що дозволяє фільтрувати та сортувати за будь-яким полем, незалежно від того, наскільки це глибоко вкладено в документі. Підтримка агрегацій та інших сучасних випадків використання, таких як пошук на основі географії, пошук графіків та пошук тексту. Запити самі по собі JSON, і тому їх легко компонувати. Дозволяє позбавитись від шаблонних об'єднуючих рядків для динамічного генерування SQL-запитів.

Модель документа відображає об’єкти у коді програми, що полегшує роботу з даними. Спеціальні запити, індексація та агрегація в режимі реального часу забезпечують потужні способи доступу та аналізу даних.

MongoDB – це розподілена база даних по своїй суті, тому її добре характеризує висока доступність, горизонтальне масштабування та географічний розподіл вбудовані та прості у використанні. MongoDB написано на C ++.

Основні можливості MongoDB:

- висока доступність завдяки вбудованій реплікації та надійності;
- горизонтальна масштабованість за допомогою вбудованих можливостей;
- має вбудовані інструменти управління для автоматизації, моніторингу та резервного копіювання.

MongoDB – це JSON подібна база даних, але заснована на специфікації BSON. В основі бази лежить документ в контексті JavaScript це звичайний об'єкт, він же є основним будівельним блоком.

У MongoDB є підтримка індексів, які прискорюють пошук. Система масштабується горизонтально через механізм шардингу і може працювати в розподіленому режимі, Також підтримується реплікація (зберігання декількох копій даних).

2.3.3. Nedb

Nedb (англ. Node.JS Embedded Database) – вбудовувана база даних для Node.JS, реалізує підмножину MongoDB API. Це невибаглива до ресурсів комп’ютера NoSQL СКБД написана на чистому JavaScript та знаходиться у відкритому доступі, не має бінарных залежностей і, окрім NodeJS, може бути використаною в NW.js та в Electron або напряму у браузері [7].

NeDB забезпечує зберігання даних у простому файлі на диску в json-формату, цей файл виконує роль на колекції у MongoDB.

Особливості:

- Підтримує основні команди MongoDB.
- Вбудовується у застосунок та здатна працювати локально.
- Підтримує індексацію для пришвидшення запитів, також підтримує індексацію по внутрішнім полям документу.
- Зберігає дані у вигляді файлів.
- З метою підвищення продуктивності NeDB використовує так званий "append-only" формат, який має на увазі, що команди

поновлення (update) або видалення (delete) записуються в кінець файлу. Так само в цілях швидкодії NeDB автоматично переводить файл з даними в однострокововий формат при завантаженні або створенні.

- Підтримує основні типи даних, що використовуються у JavaScript.
- Дозволяє налаштувати процес серіалізації та десеріалізації даних таким чином щоб вони не зберігались у файловій системі в не захищеному вигляді.

2.3.4. SQLite

SQLite – це легка і проста у налаштуванні реляційна база даних, яка легко інтегрується в різні типи пристроїв, включаючи портативні комп'ютери. Використовується для збереження даних у програмах. Наприклад, Mozilla Firefox використовує базу даних SQLite, щоб зберігати історію переглядів, закладки та інше. Сирцевий код SQLite знаходиться у відкритому доступі на GitHub тому її можливо використовувати з будь-якою метою, в тому числі і комерційною чи приватною.

На відміну від більшості звичних SQL баз даних, у SQLite немає окремого процесу, що відповідає за сервер бази даних, запис та зчитування даних відбувається безпосередньо в звичайні файли диска. Повна база даних SQL з кількома таблицями, індексами, тригерами та переглядами знаходиться в одному файлі. Формат файлу бази даних є кросплатформенним. Ці функції роблять SQLite популярним вибором як формат файлу додатків.

Кілька процесів або потоків можуть одночасно здійснювати зчитування даних з однієї бази. Запис в базу можна здійснити тільки в тому випадку, коли жодних інших запитів у цей час не обслуговується; інакше спроба запису викликає помилку. Можливий варіант автоматичного повторення спроб запису протягом заданого інтервалу часу.

Клієнтська частина працює з командного рядка, і дозволяє звертатися до файлу БД на основі типових функцій ОС.

Особливості:

- ACID транзакції;
- встановлення без додаткових конфігурації;
- реалізує значну частину стандарту SQL92;
- база даних зберігається в одному файлі на диску;
- малий фінальний розмір бази даних;
- простий, легкий у використанні API;
- доступний сирцевий код написаний на мові програмування C, який можна легко використовувати у власних проєктах;
- автономність збереження даних.

2.3.5. Висновки

Таблиця 3

Порівняння систем керування базами даних PostgreSQL, MongoDB, SQLite, Nedb

Назва	PostgreSQL	MongoDB	SQLite	Nedb
Опис	Серверна	Серверна	Вбудована	Вбудована
Модель даних	Реляційна	Документна	Реляційна	Документна
Локальне розташування	Ні	Ні	Так	Так
Схема даних	Визначена	Довільна	Визначена	Довільна
Зовнішні ключі	Так	Ні	Так	Ні
Транзакції	Так	Так	Так	Ні
Паралелізм	Так	Так	Так	Так

Задля використання локально на пристрої та відносну простоту реалізації, але за достатню швидкодію для процесів що відбуватимуться, було обрано базу даних Nedb, як досить зручну нереляційну базу даних достатню для подібних цілей, NeDB не призначений для того, щоб замінити великомасштабні бази даних, такі як MongoDB. Проте, база працює досить швидко на очікуваних наборах даних, особливо після індексації.

2.4. Вибір засобу для зберігання файлів

2.4.1. Локальне сховище у файловій системі

Для зберігання файлів найпростіше використовувати власне файлову систему комп’ютера, але це може бути небезпечно. Застосунок здатний підтримувати потокове збереження та читання звичайних файлів. Проте це досить складно реалізувати у випадку розподіленого сховища, такий варіант створить потребу вчасно синхронізувати дані між серверами, а також робити багато перевірок при запиті вже завантажених даних. Також це сховище буде доступно лише тоді коли запущено додаток.

Особливості:

- проста реалізація;
- відсутність залежностей від різноманітних модулів.

2.4.2. Minio

MinIO – це високопродуктивна розподілена система зберігання об'єктів. Він визначений програмним забезпеченням, працює на стандартному обладнання та є 100% відкритим кодом за ліцензією Apache V2. MinIO відрізняється тим, що він був розроблений з моменту створення як стандарт у приватному сховищі хмарних об'єктів. Оскільки MinIO призначений для обслуговування лише об'єктів, одношарова архітектура досягає всіх необхідних функцій без компромісів. В результаті виходить хмарний об’єктний сервер, який одночасно є ефективним, масштабованим та легким. Незважаючи на те, що MinIO переважає у використанні

традиційних об'єктів для зберігання даних, таких як вторинне зберігання, відновлення аварій та архівація, він унікальний у подоланні приватних хмарних проблем, пов’язаних із машинним навчанням, аналітикою та робочим навантаженням у хмарних додатках.

Особливості:

- з коробки підтримує розподілене зберігання даних, автоматично синхронізуючи файли між серверами [8];
- здатне працювати окремим процесом в операційній системі;
- здатне ідентифікувати користувачів які запитують дані у нього;
- детальні налаштування того хто саме, та які саме типи файлів здатен завантажувати.

2.4.3. Висновок

Таблиця 4

Порівняння засобів для зберігання даних у файловій системі та в Minio

Назва	Файлова система	Minio
Локальне розташування	Так	Одна з опцій
Робота у фоновому режимі	Так	Так
Розподілення сховища	Ні	Так
Шифрування вмісту	Так	Так

Отже за сукупністю необхідних системі можливостей простіше скористатися готовим та інструментом – Minio, його значно зручніше налаштовувати та підтримувати його працездатність.

2.5. Середовище розробки

2.5.1. WebStorm

WebStorm – середовище розробки від компанії JetBrains. Розроблене для створення програмних продуктів на мові JavaScript та похідних від неї фреймворках та бібліотеках. Середовище розробки надає широкий вибір інструментів для програмування як клієнтської, так і серверної частини додатків, містить велику кількість вбудованих плагінів для ефективної розробки веб-додатків.

Варто зазначити, що безкоштовно користуватись продукцією компанії JetBrains можна лише з обмеженим набором інструментів, але для студентів компанія надає можливість оформлення безкоштовної підписки для навчальних цілей. WebStorm самостійно враховує логіку підключених бібліотек та модулів, що робить його особливо зручним для розробки на JavaScript та похідних технологіях; найзручнішими можливостями є перевірка на помилки та підсвічування коду.

2.5.2. VSCode

Visual Studio Code – це легкий, але потужний редактор вихідного коду, який працює на робочому столі та доступний для усіх популярних зараз операційних систем. Розроблений та підтримується спільнотою GitHub, сирцевий код знаходиться у відкритому доступі. Написаний на платформі Electron. Має велику кількість доступних до завантаження доповнень, що дозволяють використовувати цей редактор для роботи з будь-якою мовою програмування.

2.5.3. Висновок

Таблиця 5

Порівняння середовищ розробки WebStorm та VSCode

Назва	WebStorm	VSCode
Підтримка обраної мови програмування	Так	Так
Підтримка автоматичної компіляції коду	Так	Так, необхідно попереднє налаштування
Аналіз коду	Так	Так, за допомогою плагінів
Безкоштовний доступ	Ні, можливий безкоштовний доступ при оформленні студентської підписки	Так

Таким чином, враховуючи наявність студентської підписки, було обрано середовище розробки WebStorm від компанії JetBrains, як більш потужне для написання програм мовою TypeScript та роботи зі сторонніми бібліотеками.

2.6. Додаткові технології та бібліотеки

- minio – бібліотека, що полегшує взаємодію зі сховищем, має описані типи та підтримує потокову роботу, написана на JavaScript, також підтримує роботу з API S3 сховищ в цілому, тобто воно дозволить використовувати одним з вузлів вже існуюче сховище, наприклад на Amazon;
- busboy – бібліотека, що дозволяє отримувати дані від користувача, під час завантаження файлів та здійснити їх потокову обробку, таким чином обробник приймає запит з мережі та дозволяє обробити дані форм, використання саме цієї

бібліотеки зумовлене тим що вона здатна максимально оптимізовано використовувати ресурси комп’ютера;

- nedb – бібліотека, що повністю відповідає за базу даних Nedb, дозволяє її створити у оперативній пам’яті та виконувати запити через її API, підтримує створення індексів для пришвидшення запитів, а також налаштування серіалізації та десеріалізації даних в автоматичному режимі, також сама слідкує чи не була втрачена частина даних і чи все було коректно збережено;
- bcrypt – бібліотека, що реалізовує однойменний криптографічний алгоритм, цей алгоритм дозволяє бути впевненим що у випадку коли дані користувачів стануть доступними стороннім особам вони все одно не змогли б відновити такі важливі дані, як пароль [9];
- child_process – вбудований модуль стандартної бібліотеки Node.JS, дозволяє запускати дочірніми процесами необхідні застосунки, у даному випадку використовується для створення процесу серверу Minio та керування ним під час роботи програми;
- datagram – модуль стандартної бібліотеки NodeJS що описує взаємодію за допомогою протоколу UDP, що дозволяє обмінюватись пристроям даними знаючи лише IP адреси одне одного, а при знаходженні іншого, встановлювати вже захищений зв'язок між ними по протоколу TCP;
- jsonwebtoken – бібліотека що реалізує однойменний стандарт та дозволяє досить безпечно аутентифікувати запити від користувачів;
- express – фреймворк для мови Node.JS який дозволяє будувати API будь якого масштабу, а також для нього створено багато бібліотек для полегшення процесу розробки.

2.7. Висновки

У даному розділі я розглянув основні інструменти та технології для створення веб-застосунків. Вихідні дані були систематизовані та проаналізовані з метою подальшого використання у роботі визначеного набору інструментів, які будуть використовуватись для розробки децентралізованого сховища даних. Основною мовою розробки було вирішено обрати TypeScript, основними аргументами для цього рішення були:

- часте використання даної мови у веб-програмуванні та її пристосованість саме до цієї області розробки, а також підтримка великої кількості бібліотек та фреймворків;
- чисельна спільнота розробників;
- обрана для проєкту база даних Nedb має зручний для інтеграції з веб технологіями інтерфейс;
- в якості інтерфейсу користувача було обрано сторінку, яка відкривається у браузері користувача при запуску додатку та дозволяє взаємодіяти з системою.

3. ОПИС РОЗРОБЛЕНОГО ВЕБ-ДОДАТКУ

3.1. Загальний опис

Дана система є додатком, який встановлюється у операційну систему, працює як окремий процес та у продовж роботи створює допоміжні процеси.

Неавторизований користувач не здатний робити ніяких взаємодій, йому спочатку необхідно ввести логін і пароль, це буде логіном та паролем до його облікового запису.

Зареєстровані облікові записи мають наступні можливості:

- додавати підключення до сховищ даних або створювати своє власне;
- додавати будь-які файли, та переглядати наявні;
- переглядати статистику використання файлів;
- додавати інших користувачів та визначати їм доступ до окремих частин сховища;
- виконувати пошук по завантаженим файлам;
- виступати у ролі універсального посередника, у випадку використання якогось сховища, яке не є сервером Minio, тому що застосунок підтримує узагальнене API для роботи з такими сховищами;
- робити свій комп’ютер доступним з зовнішньої мережі, тобто при необхідності надати доступ до даних ти кому потрібно, а сервер Minio дозволяє зробити саме такі налаштування.

Додаток повинен бути здатним самостійно створювати канал з’єднання з потрібним користувачем, та зберігати його у профілі користувача.

3.2. Архітектура додатку

Для розроблення продукту було обрано однорангову архітектуру.

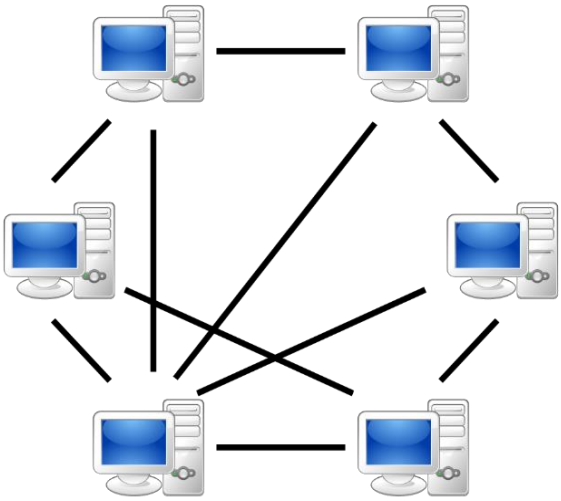


Рис. 2. Архітектура однорангової мережі

Однорангова мережа передбачає що кожен її елемент – користувач буде виконувати таку саму функцію як і інші, в даному випадку – створення з’єднання між користувачами, зберігання даних і така архітектура не передбачає якого виділеного сервера що займатиметься цими задачами, кожен користувач в певному роді буде цим сервером.

У одноранговій мережі комп'ютери в мережі рівні, при цьому кожна робоча станція забезпечує доступ до ресурсів і даних. Це простий тип мережі, де комп’ютери можуть спілкуватися один з одним і ділитися тим, що знаходиться на комп'ютері або приєднано до нього з іншими користувачами. Це також один із найпростіших типів архітектури для створення.

Основні характеристики таких мереж:

- окремі користувачі несуть відповідальність за доступ до даних та ресурсів на своїх комп’ютерах;
- операційні системи, такі як Windows XP та Windows Vista, дозволяють створити облікові записи, які будуть

використовуватися, коли інші користувачі підключаються до комп'ютера окремого користувача він буде в змозі контролювати свої ресурси;

- облікові записи, паролі та дозволи зберігаються в локальній базі даних і використовуються для визначення того, що хтось може робити під час підключення до комп'ютера.

Вони також дуже вразливі до відмови в сервісних атаках, оскільки кожен пристрій сприяє маршрутизації трафіку через мережу. Використання однорангової архітектури не виключає застосування в тій же мережі також архітектури «термінал-головний комп'ютер» або архітектури «клієнт-сервер».

P2P-мережа має ряд переваг. Наприклад, у традиційній моделі мережі клієнт-сервер, якщо сервер виходить з ладу, він може взяти з собою всю мережу. Але в P2P, якщо один пристрій виходить з ладу, інші мережі можуть допомогти зібрати слабкість. Вони також допомагають гарантувати, що мережевий трафік не є вузьким місцем на одному пристрої, оскільки обробка трафіку розподіляється на багатьох комп’ютерах.

Зберігання даних відбувається шляхом опитування усіх учасників мережі на поточний статус збереження, тобто документ не вважатиметься збереженим доки він не зберігся на кожному пристрої в мережі.

При отриманні та завантаженні даних з зовнішніх джерел вони автоматично розподіляються між усіма учасниками задля досягненнях синхронізації поточного стану файлів на усіх пристроях у мережі.

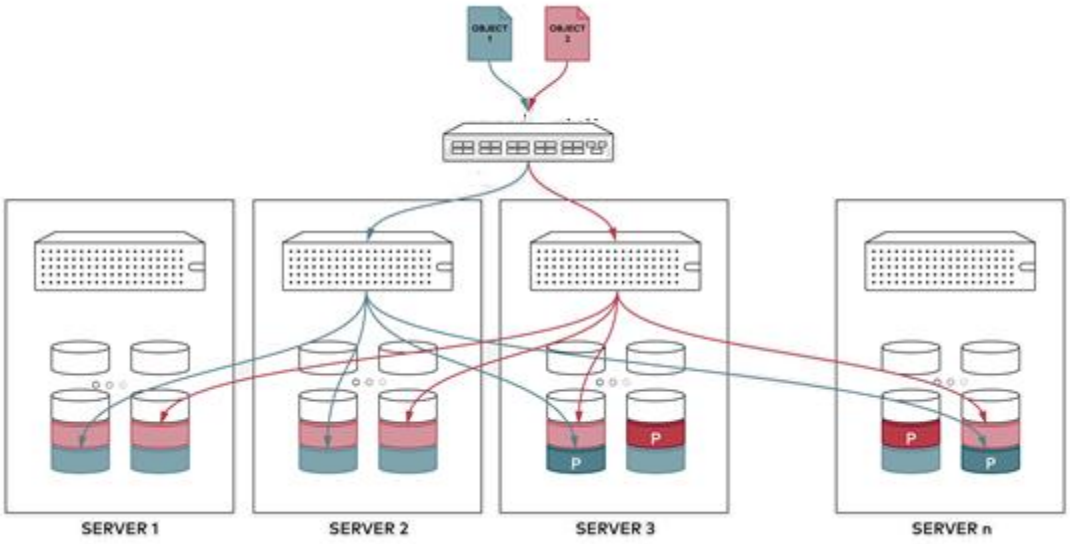


Рис. 3. Алгоритм синхронізації даних між вузлами

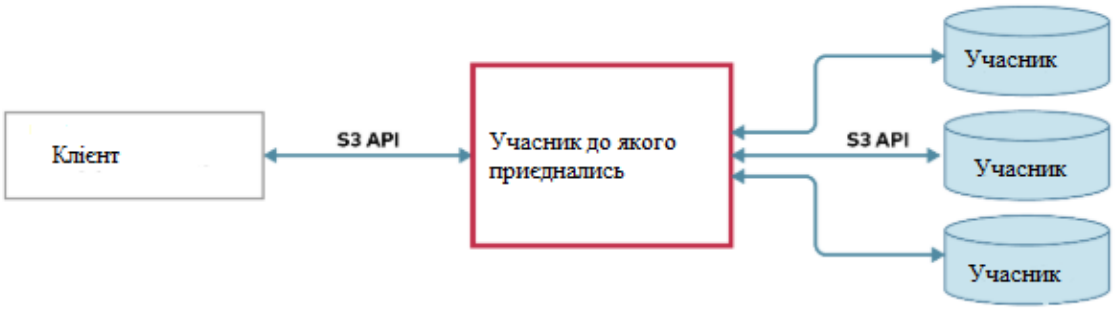


Рис. 4. Алгоритм синхронізації даних при взаємодії зі стороннім користувачем

Таким чином система автоматично забезпечує наявність реплікації даних на усіх пристроях учасниках мережі.

3.3. Функціональні вимоги

Додаток має наступні функціональні вимоги:

- забезпечити можливість користувачеві власноруч створити та запустити власне сховище, доступ до якого спочатку матиме лише він, але зможе надавати доступ кому забажає;
- реалізація власне функціоналу сховища у вигляді спеціальної утиліти, що запускається на пристрої, виділяє для своїх потреб

необхідну кількість пам’яті на пристрої та використовує її як єдиний блок, всередині він може мати безліч різноманітних даних, але для інших програм та при просто пошуку серед файлів усе сховище включно зі змістом представляє собою єдиний файл, який при потребі можна максимально просто скопіювати, або при потребі видалити;

- використання системи, що складається з одного файлу також забезпечить можливість легко зашифрувати такий файл і зберігати свої дані ще й у недоступному без необхідних для розшифрування даних вигляді для перегляду, навіть у разі якщо хтось сторонній все ж отримав до них фізичний доступ;
- реалізація реплікації даних при необхідності, задля синхронізованого зберігання не лише на одному пристрої;
- підтримка актуальних на даний момент форматів даних;
- використання найбільш оптимізованих, швидких та надійних засобів для зберігання та обробки даних;
- реалізація можливості віддаленого доступу до даних через посилання на запущений на пристрої сервіс-сховище, можливості поширення та налаштування доступу до даних іншим користувачам;
- створення власне системи для використання описаних вище засобів через узагальнений API інтерфейс.

3.4. Опис модулів програмного забезпечення

У структурі розробленого додатку можна чітко виділити таки ключові модулі:

- модуль авторизації та реєстрації;
- модуль взаємодії з підключеними сховищами;
- модуль локальних сховищ;
- модуль взаємодії з базою даних;
- модуль обробки завантажуваних даних.

3.4.1. Модуль авторизації та реєстрації

Сутність користувача складається з наступних полів:

- login – ім’я користувача за яким він аутентифікується та його можна знайти;
- password – пароль користувача, захищений за допомогою алгоритму S3;
- _id – унікальний ідентифікатор у мережі.

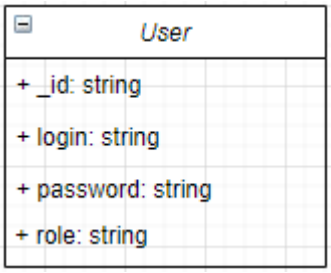


Рис. 5. Схема сутності користувача

Після введення даних, виконується перевірка правильності вводу логіну та паролю і якщо все сходиться то він продовжує роботу, якщо ж логіну не було знайдено створюється новий користувач, а якщо пароль було введено неправильно то про це повідомляється окремо. Авторизація здійснюється при кожному запиті користувача до системи, окрім власне самого запиту на авторизацію задля здійснення контролю доступу. Авторизація здійснюється за допомогою JWT токена.

3.4.2. Модуль взаємодії з підключеними сховищами

Сутність з’єднання має в собі такі дані:

- name – ім’я з’єднання щоб можна було його знайти у списку;
- type – вказує на те чи це локальне чи віддалене з’єднання та за якими інструкціями з ним надалі взаємодіяти;
- userId – вказує на те якому саме користувачеві належить це з’єднання, для можливості створювати користувачів та надавати їм доступ лише до певних частин;

- options: – об’єкт що зберігає у собі параметри які необхідні для того щоб здійснити з’єднання;
- _id – унікальний ідентифікатор з’єднання.

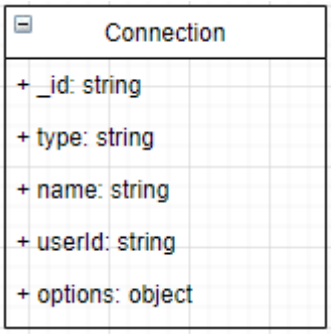


Рис. 6. Схема сутності з'єднання

Також цей модуль не лише створює записи у базі даних, а ще й налаштовує потрібним чином взаємодію з ними, надає методи для потокового завантаження, перегляду списку файлів, дозволяє робити файли доступними іншим користувачам, проводить аналіз їх використання. Обидві структури – user, connection індексуються по полям _id задля пришвидшення запитів.

3.4.3. Модуль локальних сховищ

Модуль що використовує стандартну бібліотеку Node.JS, а саме модуль child_process, який дозволяє створювати дочірні процеси та підтримувати їхню роботу, відповідає за коректний старт та завершення процесу локально серверу Minio, та за обробку подій які стаються з дочірнім процесом під час роботи, наприклад неочікуваний збій у роботі серверу, таким чином даний модуль створює зручну абстракцію для взаємодії запуску допоміжних утиліт в операційній системі [10].

Також це модуль надає абстракцію для взаємодії не лише з локальним сховищем, дозволяє переглядати наявний зміст, дає можливість доступу до наявних файлів та можливості додавання нових, але для цього вони працюють разом з модулем завантаження даних, тому що з завданням

прийняти дані та перевірити їх правильність – задача модуля завантаження, а власне зберігання виконує модуль сховищ.

Сутність файлу у сховища має наступну структуру:

- name – ім’я файлу, унікальне, проте якщо буде завантажено файл, але така назва вже існує, то старий файл буде перезаписано;
- etag – хеш-сума, один з заголовків http, з якими не взаємодіє напряму користувач, але взаємодіє його браузер, є результатом хеш-функції виконаної над даними, щоб отримати їх унікальний ідентифікатор та при повторному запиті даних не завантажувати їх ще раз, а взяти з локального сховища, якщо значення параметру etag співпадає;
- lastModified – також допоміжний параметр що забезпечує можливість кешування даних, є строковим записом останньої дати коли над файлом було здійснено маніпуляції, таким чином можна визначити чи актуальна версія файлу вже наявна на пристрої чи потрібно запросити нову;
- size – параметр що вказує на розмір файлу у байтах, відіграє роль при потоковому отриманні даних за сховища, тому що спочатку невідомо кінцевий розмір файлу, тому браузеру необхідно виділити під нього певний обсяг пам’яті.

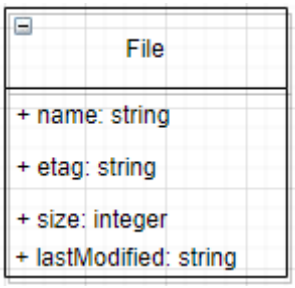


Рис. 7. Схема сутності файлу

3.4.4. Модуль взаємодії з базою даних

Відповідає за початкове налаштування, створення необхідних директорій, необхідних для бази даних, та описує клас для взаємодії з тією чи іншою сутністю, надає більше високорівневі методи для запитів до бази даних ніж це було в модулі Nedb спочатку та додає в нього повноцінну асинхронність. Виконує завантаження даних при запуску додатку та коректне збереження даних при його завершенні.

3.4.5. Модуль обробки завантажуваних даних

Модуль що працює з потоками даних у вигляді байтів інформації, які в сумі представляють собою повноцінний файл, але обробка маленькими частинами дозволяє значно підвищити пропускну здатність та продуктивність в цілому. Підтримує одночасне завантаження багатьох файлів [11].

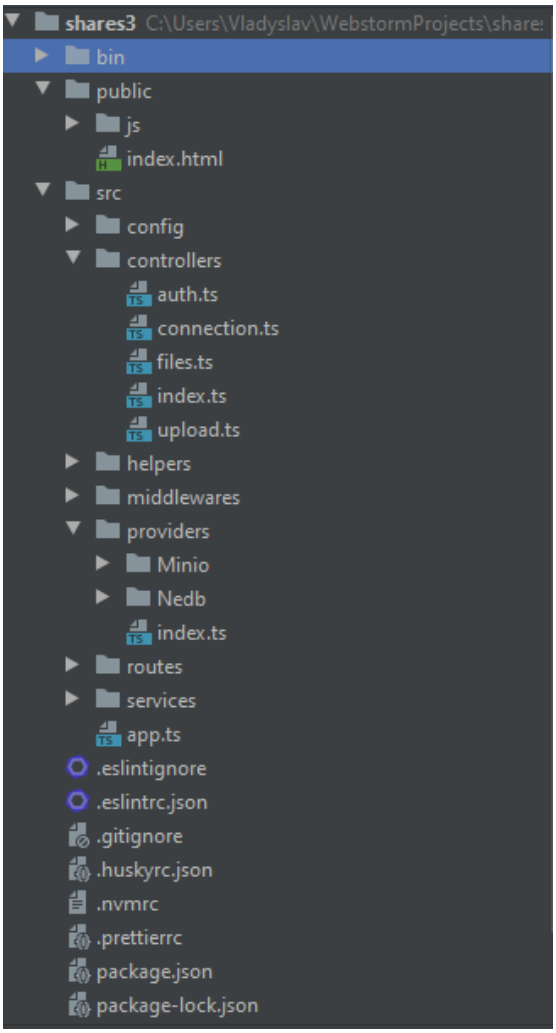


Рис. 8. Загальна структура додатку

4. АНАЛІЗ РОЗРОБЛЕНОГО ДОДАТКУ

Систему розроблено за стандартами шаблону розподіленої архітектури. Робочий потік застосунку було побудовано згідно з сучасними стандартами розроблення інтерфейсів веб додатків.

Робочий потік складається з наступних кроків:

- авторизація у додатку;
- завантаження списку зєднань;
- посилається запит на отримання списку доступних файлів по обраному з’єднанню;
- відображення списку файлів, можливості додати нові та видаляти або змінювати ті що вже є.

4.1. Робочий потік додатку

На головній сторінці міститься форма для введення логіну та паролю користувача, також виконує роль форми для реєстрації нових, якщо за введеним логіном не знайшлося користувача.

Рис. 9. Сторінка автентифікації

При успішній авторизації у відповідь на запит приходить токен, який необхідний для подальшої взаємодії та здійснення запитів, інакше система не дозволить доступ до даних.

Після того як користувач авторизувався робиться запит на отримання списку під’єднань доступних даному користувачу.

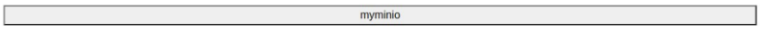


Рис. 10. Сторінка вибору необхідного з'єднання серед доступних

Після вибору необхідного сховища відображається список доступних у ньому файлів.

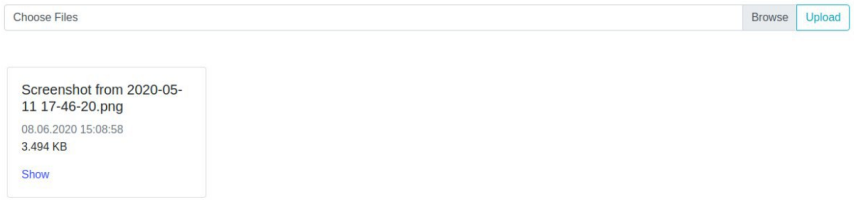


Рис. 11. Сторінка перегляду завантаженого вмісту

Також доступна форма для завантаження одночасно багатьох файлів, при натисканні на кнопку “Show” відбувається відкриття нової вкладки у браузері то переадресація на сторінку з доступним переглядом даних безпосередньо у браузері або завантаженні даних на свій пристрій в залежності від їх формату.

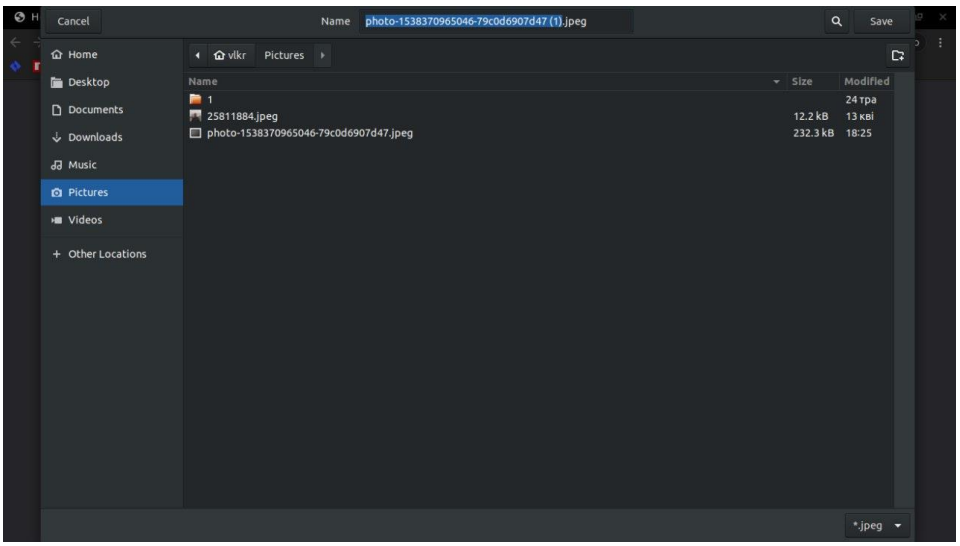


Рис. 12. Сторінка збереження вмісту на свій пристрій

Це посилання за замовчуванням діє один тиждень, тобто його можна передати кому потрібно для перегляду і після того як пройде тиждень цей файл просто стане недоступним по даному посиланню, проте залишиться у сховищі, ці налаштування можна змінювати в залежності від поставленої задачі.

4.2. Тестування засобу

Контроль за якістю є постійною та важливою частиною життєвого циклу розроблення та підтримки програмного забезпечення та входить в кожен етап створення додатку, як завершальна частина перед затвердженням імплементації функціональності.

За відповідність необхідним можливостям, прописаних у постановці задачі, відповідають модульні тести, що покривають основний функціонал, та перевіряють його при оновленнях, щоб не зламались основні функції.

Так як проєкт написаний мовою TypeScript то компілятор на своєму рівні сам допомагає написати такий програмний код, що не матиме проблем при роботі з різними типами даних, а отже і повинен працювати так як це задано інструкціями у коді.

Тестування направлене на виявлення помилок наступного типу:

- помилки пошуку у базі даних;
- помилки у неправильному динамічному визначенні з’єднання яке використається зараз користувачем;
- помилки при завантаженні одного декількох файлів різних форматів;
- помилки при зчитуванні даних;
- помилки при встановленні зв’язку між користувачами;
- помилки під час запуску та початкового налаштування процесу що відповідає за локальне сховище.

4.3. Порівняння розробки з існуючими аналогами

Розроблений додаток відповідає усім визначеним вимогам, а саме:

- можливість створення локального сховища даних;
- доступність цього сховища з мережі;
- підтримка графічного інтерфейсу користувача;
- можливість під’єднання до різноманітних за реалізацією та позиціонуванням сервісів для зберігання даних, які використовують S3 подібну архітектуру;
- зберігання усіх даних додатку локально у спеціально відведеному для нього місці, незалежно від операційної системи та можливість безпечної взаємодії з ним;
- потокова обробка даних на вході та на виході;
- перевірка даних на відповідність вказаному формату;
- можливість працювати одному і тому ж сховищу одночасно на кількох пристроях з автоматично синхронізацією даних між ними.

4.4. Рекомендації щодо подальшого вдосконалення

Розроблений засіб можна використовувати як при розробках програмних проєктів, та забезпечується можливістю розкрити сховище локально та віддалено. Основною задачею що на нього поставлено це забезпечення зберігання будь яких своїх даних та можливості доступу до них коли це потрібно.

Проте залишилось багато частин які можна було б покращити, наприклад додати можливість використання ще й розподіленої між групами користувачів базами даних, це також дозволить гарантувати достовірність даних, що зберігаються, перевіряючи їх хеш-суми.

Також можливе доопрацювання графічного інтерфейсу користувача для підтримки та виведення більшої кількості необхідної інформації та більшої кількості налаштувань доступу.

ВИСНОВКИ

Метою дипломного проєкту було створення децентралізованого сховища даних, яке б використовувало не локальну файлову систему комп’ютера, а більш сучасні та універсальні засоби.

Після проведення аналізу існуючих програмних рішень у даній сфері було вирішено розробити систему у вигляді програми для комп’ютеру, яка взаємодіє з користувачем через сторінку у браузері, такий підхід мінімізуватиме помилки користувачів та дозволяє їм виконувати усі вище описані дії.

Розроблений додаток надає:

- можливість зареєструватись або увійти у свій акаунт;
- можливість оперувати доступними та запускати власні сховища даних;
- з’єднуватись з іншими учасниками для спільного та більш надійного зберігання даних.

Розробку програмного забезпечення виконано у повному обсязі та у відповідності до сформованих вимог. Тестування веб-застосунку виконано за затвердженою програмою та методикою тестування.

Використання розробленого додатку забезпечить користувачам зручний та надійний сервіс для зберігання даних.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Object Storage [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Object_storage Дата доступу: травень 2020. Назва з екрану
2. SSH VPN Tunnel [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/331348/#t5 Дата доступу: травень 2020. Назва з екрану
3. Cloud Storage [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Cloud_storage Дата доступу: травень 2020. Назва з екрану
4. How to use AWS SDK for Javascript with MinIO Server [Електронний ресурс]. – Режим доступу: https://docs.min.io/docs/how-to-use-aws-sdk-for-javascript-with-minio-server.html Дата доступу: травень 2020. Назва з екрану
5. Как управлять и редактировать метаданными фотографий и файлов [Електронний ресурс]. – Режим доступу: https://webznam.ru/blog/metadannye_fajlov_fotografij/2015-04-01-135 Дата доступу: травень 2020. Назва з екрану
6. Validation And Serialization [Електронний ресурс]. – Режим доступу: https://www.fastify.io/docs/latest/Validation-and-Serialization/ Дата доступу: травень 2020. Назва з екрану
7. NeDB: аналог SQLite для NodeJS [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/301916/ Дата доступу: травень 2020. Назва з екрану
8. Distributed MinIO Quickstart Guide [Електронний ресурс]. – Режим доступу: https://docs.min.io/docs/distributed-minio-quickstart-guide.html Дата доступу: травень 2020. Назва з екрану

9. Bcrypt [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Bcrypt Дата доступу: травень 2020. Назва з екрану
10. Asynchronous Process Creation [Електронний ресурс]. – Режим доступу: https://nodejs.org/api/child_process.html#child_process_child_process_spawn_command_args_options Дата доступу: травень 2020. Назва з екрану
11. Busboy [Електронний ресурс]. – Режим доступу: https://github.com/mscdex/busboy Дата доступу: травень 2020. Назва з екрану.

ДОДАТКИ

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2019 р.

ДЕЦЕНТРАЛІЗОВАНА СИСТЕМА ДЛЯ ЗБЕРІГАННЯ ТА ПОШИРЕННЯ ДАНИХ НА ОСНОВІ ТЕХНОЛОГІЇ S3

Програма та методика тестування

ДП.045440-04-51

«ПОГОДЖЕНО»
Керівник проєкту:

_____ Юрій БУХТІЯРОВ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Владислав КИРИЧЕНКО

Київ 2019

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Додаток для децентралізованого зберігання поширення даних на основі технології S3, серверна частина якого розроблена з використанням Node.JS та TypeScript, а клієнтська – з використанням фреймворку Vue.JS.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1) відповідність функціональних можливостей, реалізованих у додатку до заявлених;
2) коректність поведінки додатку;
3) забезпечення належного рівня безпеки даних;
4) зручність роботи з додатком.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування проводитиметься на рівні «системного тестування», тобто буде перевірено кожен модуль розробленого програмного продукту та зв’язки між ними. Для проведення тестування було обрано метод White Box Testing. При такому методі перевіряється як поведінка програмного продукту, так і код.

Використовуються наступні методи:

1) для проведення функціонального тестування Critical path test (тестування критичного шляху);
2) для тестування продуктивності програмного забезпечення load testing (навантажувальне тестування) та stress testing (стресс-тестування);
3) тестування інтерфейсу.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Коректність роботи вебзастосунку тестується з використанням таких методів та інструментів:

1) введенням недопустимих значень в поля, що можна редагувати;
2) ручного тестування відповідності реалізованих функціональних вимог до заявлених;
3) тестування при максимальному навантаженні за допомогою утиліти Apache JMeter;
4) тестування стабільності роботи при різних умовах за допомогою утиліти Apache JMeter;
5) тестування веб-застосунку в різних браузерах;
6) тестування зручності користувацького інтерфейсу опитуванням потенційних користувачів.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

ДЕЦЕНТРАЛІЗОВАНА СИСТЕМА ДЛЯ ЗБЕРІГАННЯ ТА ПОШИРЕННЯ ДАНИХ НА ОСНОВІ ТЕХНОЛОГІЇ S3

Керівництво користувача

ДП.045440-04-51

«ПОГОДЖЕНО»
Керівник проєкту:

_____ Юрій БУХТІЯРОВ

Нормоконтроль:
_____ Микола ОНАЙ

Виконавець:
_____ Владислав КИРИЧЕНКО

2020

ЗМІСТ

1. Опис структури додатку

Додаток для децентралізованого зберігання та поширення даних являє собою веб-додаток, розроблений як SPA (односторінковий додаток). Мова інтерфейсу – англійська. Структура сторінки складається з навігаційної панелі, що розташована зверху, та основної частини сторінки нижче. Навігаційна панель містить наступні елементи:

- вкладка «Connections», яка веде на основну сторінку;
- вкладка «Settings», яка веде до основних налаштувань та дозволяє виставити обмеження на час доступності посилань та налаштування локальних сховищ;
- вкладка «Statistics», яка ведена сторінку де можна отримати інформацію про те скільки сховищ наразі підключено до мережі та завантаженість операційної системи;
- кнопка «Disconnect» для негайного відключення від спільної мережі сховищ;
- кнопка «Log Out» для відключення від спільної мережі сховищ, скидання налаштувань доступності файлів за посиланням та вимкнення усіх локальних сховищ.

Навігаційна панель наявна завжди та виділяє відкриту у даний момент вкладку.

При роботі сторінка додатку складається з наступних розділів:

- розділ «Login» – початковий екран, після введення даних перенаправляє у розділ «Connections»;
- розділ «Connections» – розділ за замовчуванням, користувач одразу перейде у нього, якщо перед цим вже пройшов авторизацію, у цьому розділі також відбувається додавання нових підключень, при виборі якогось з доступних підключень відкриває розділ з файлами, що доступні у ньому;

- розділ «Settings» для налаштувань;
- розділ «Files» для перегляду доступних файлів та взаємодії з ними, містить кнопку «Upload», котра перенаправляє на сторінку завантаження;
- розділ «Uploading» дозволяє завантажувати файли безпосередньо у сховище, дозволяє завантажувати по декілька файлів та підтримує Drag-and-drop – перетягування необхідних файлів просто у простір який займає сторінка;
- розділ «Statistics» для перегляду додаткової інформації про поточний стан роботи додатку.

Перехід між сторінками відбувається за натисканням відповідних кнопок на навігаційній панелі.

2. Процедура авторизації користувача

Сторінка «Login» відкривається за змовчуванням для неавторизованих користувачів (рис. 1). Сторінка містить форму для вводу необхідних для авторизації даних: юзернейма та пароля. Після введення коректних даних та натиснення кнопки «Submit», користувач увійде в систему.

Рис. 1. Сторінка «Login»

Після успішного здійснення входу у систему користувача перенаправить у розділ «Connections». А при наступному вході на сторінку протягом одного дня не буде запитуватись логін та пароль упродовж одного дня за замовчуванням.

3. Процедура додавання підключених сховищ

Сторінка «Connections» (рис. 2) містить список що відображає усі додані та підключенні сховища, форму для вводу необхідних даних для додавання нового, та можливість поєднати потрібні для використання у розподіленому вигляді. Після введення коректних даних та натиснення кнопки «Connect», налаштування буде додано.

Рис. 2. Сторінка «Connections»

За замовчуванням при додаванні сховища яке відсилається на локальну адресу виконується перевірка його робити, та у разі якщо воно не працює його буде увімкнуто та додано до спільної мережі сховищ.

4. Розділ «Files». Керування доступними файлами

Після успішного підключення до спільного сховища або при натисканні у розділі «Connections» на потрібне, користувач потрапляє у розділ «Files» (рис. 3). У даному розділі можна переглянути доступні файли та провести певні дії над ними. При натисканні на кнопку «Show», відбувається переадресація на нову вкладку для перегляду чи завантаження ораного файлу, в залежності від типу цього файлу (рис. 4).

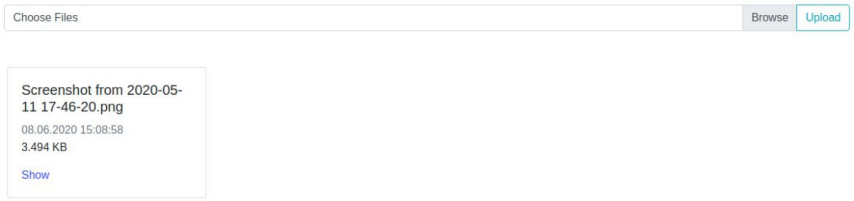


Рис. 3. Сторінка «Files»

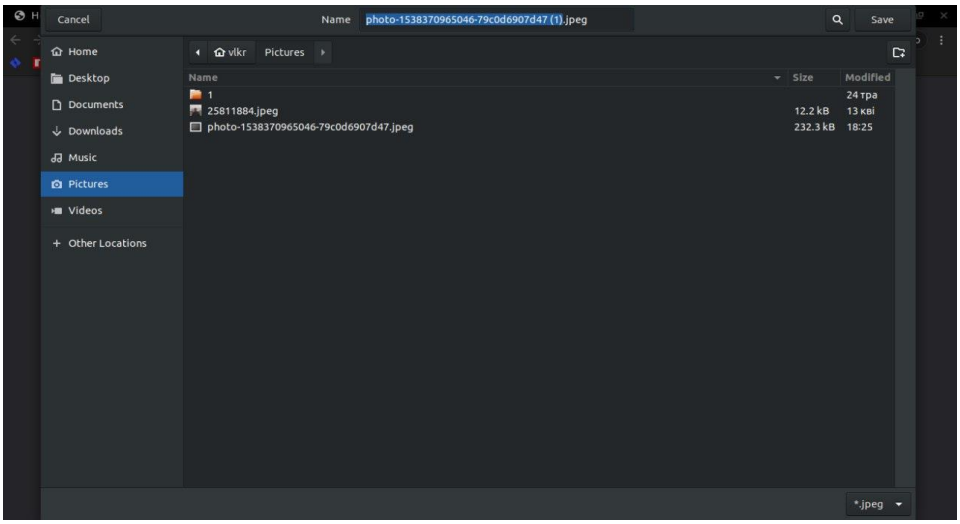
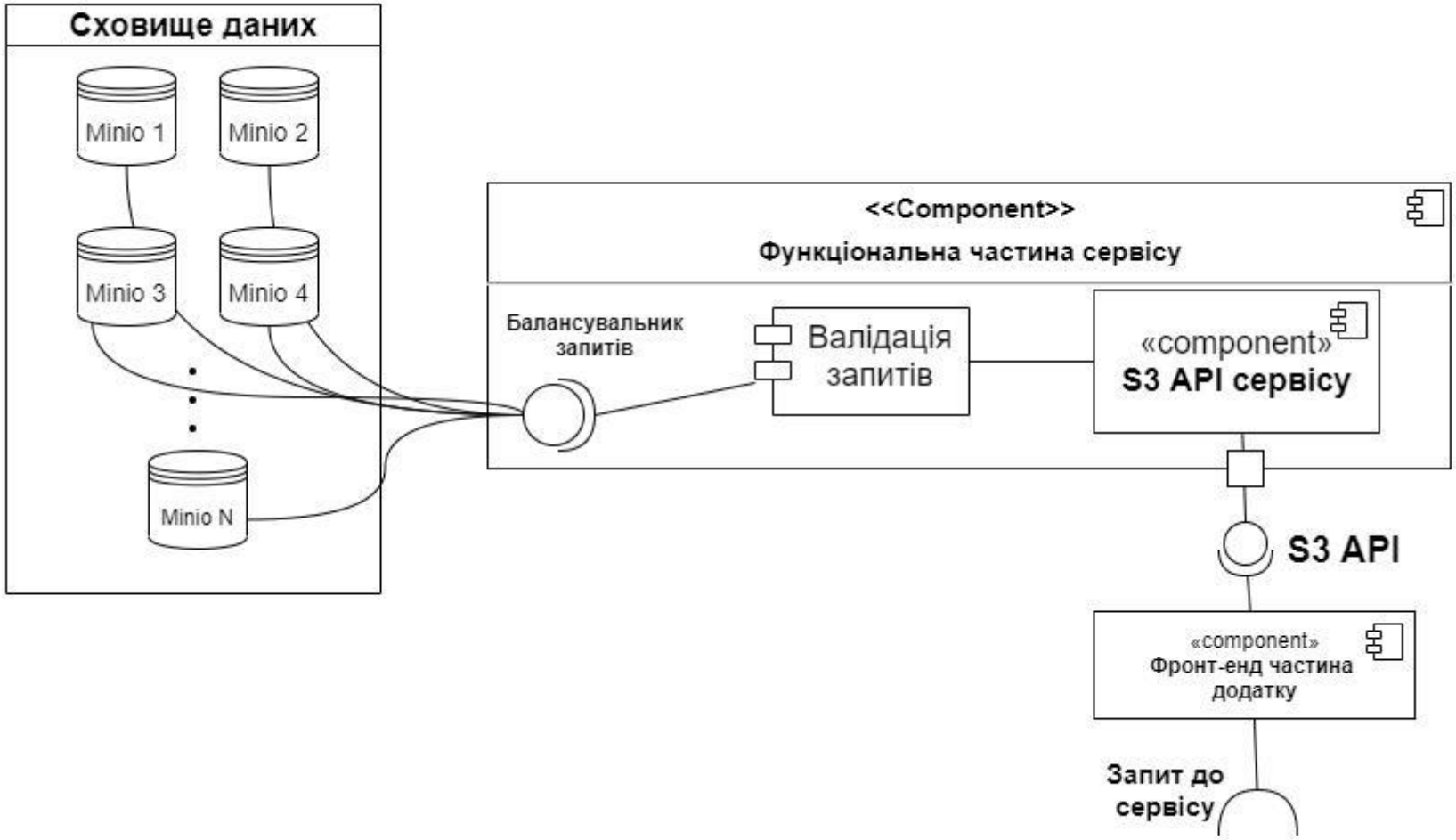


Рис. 4. Модальне вікно для збереження файлу

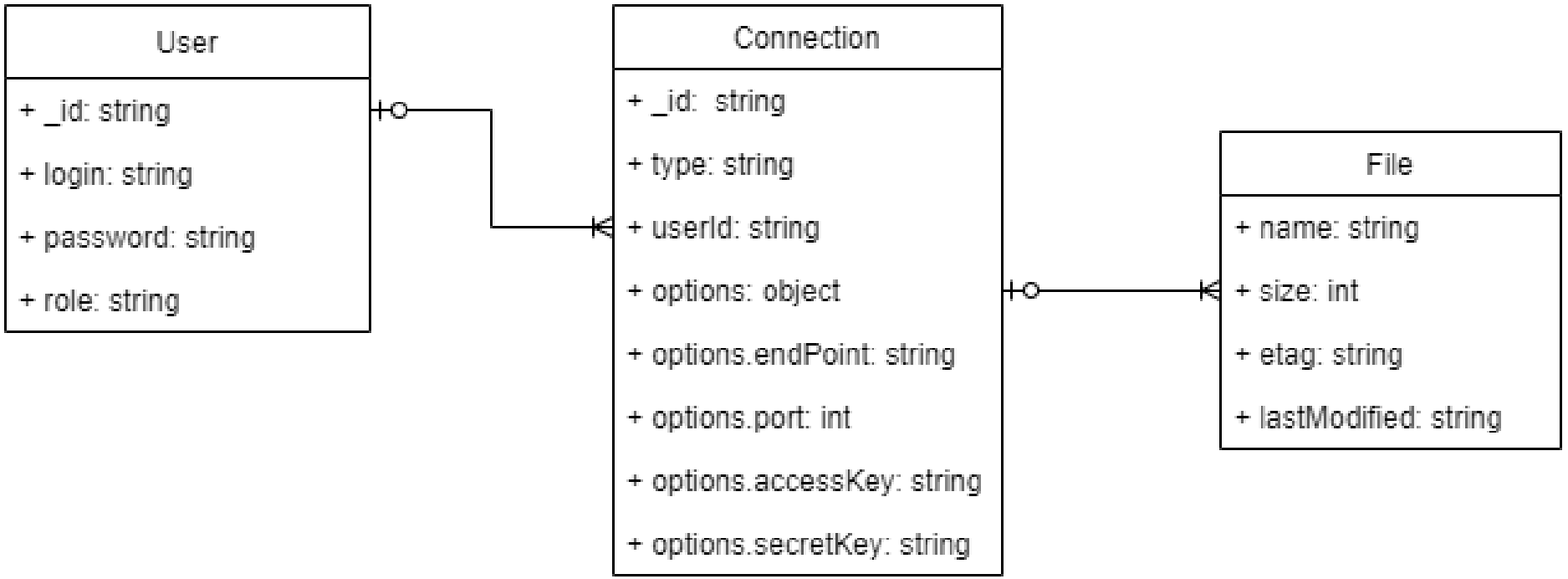
Додаток 1

Копія графічних матеріалів



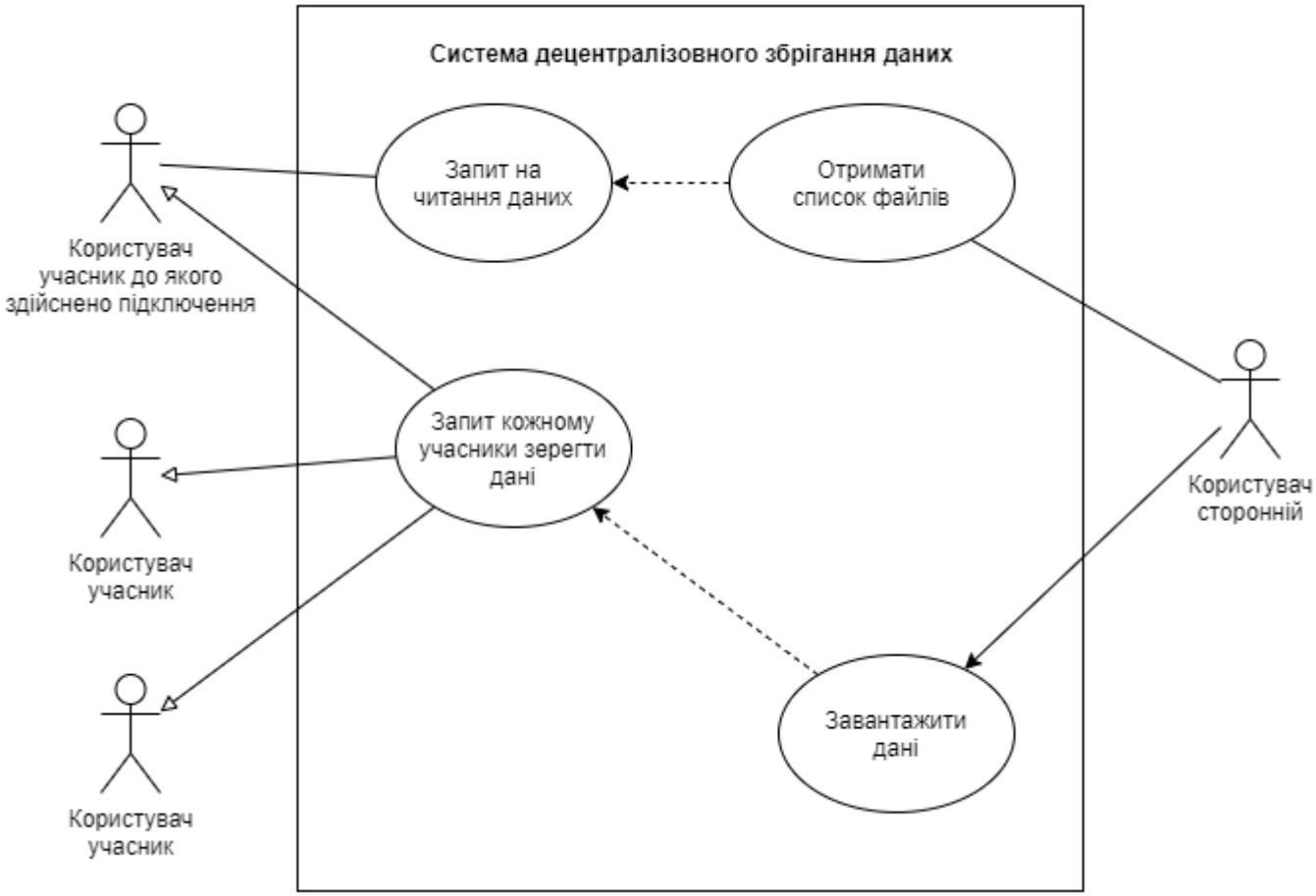
ДП.045440-06-99

Децентралізована система для зберігання та поширення даних на основі технології S3. Діаграма компонентів системи. Архітектура системи



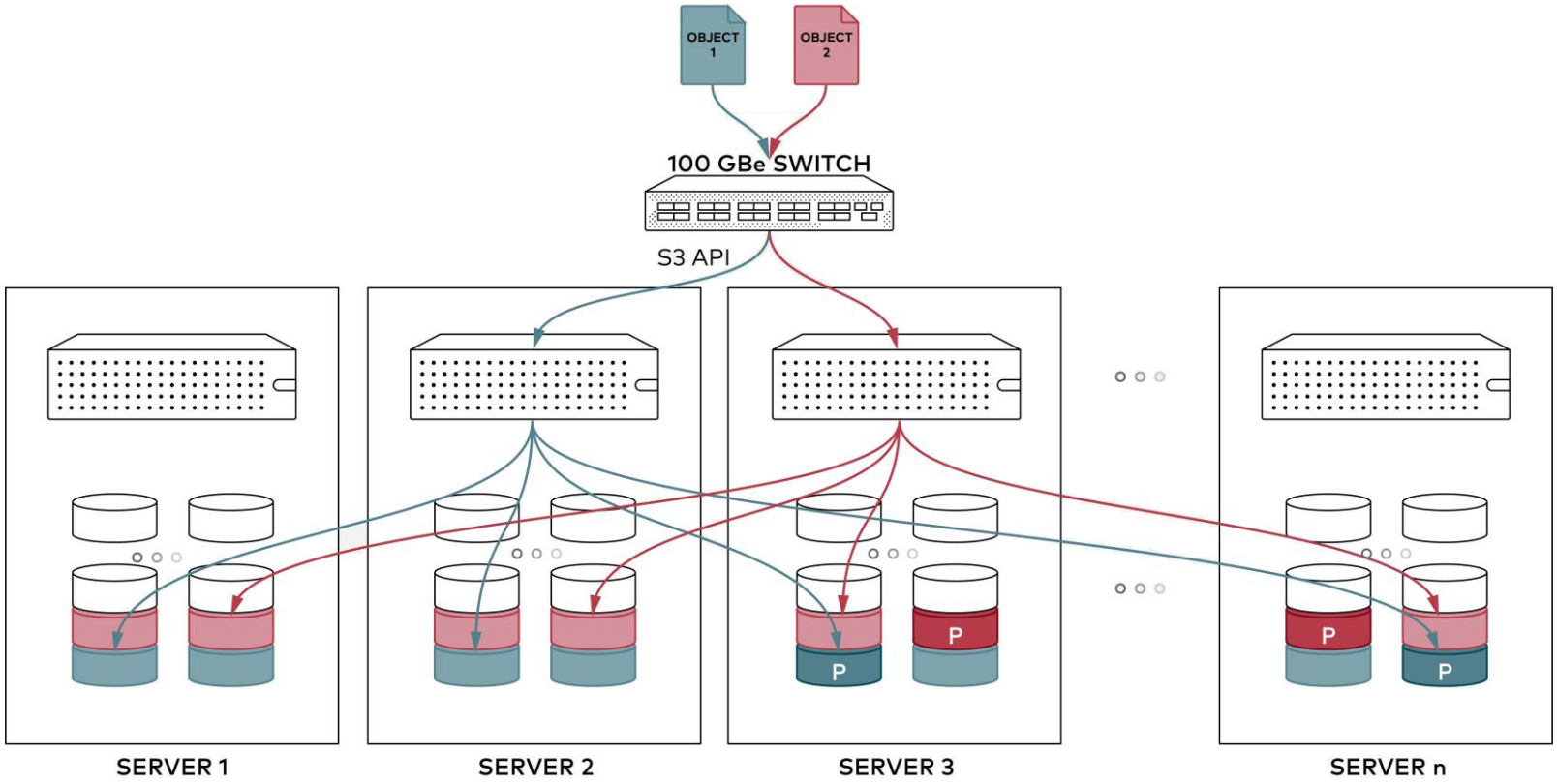
ДП.045440-07-99

Децентралізована система для зберігання та поширення даних на основі технології S3. База даних системи. Схема бази даних



Кириченко В.А., група КП-62

Кириченко В.А., група КП-62



Кириченко В.А., група КП-62

Додаток 2

Лістинг програми

Лістинг 1. Модуль для роботи зі сховищем даних

```
import { Stream } from 'stream'
import { Client, BucketItem } from 'minio'
import { IConnection } from '../../Nedb'
import { DownloadSteam } from './stream'
import { LocalStorageService } from '../../../services'
import { MinioOptions, Item } from '../../../interfaces'

export class Minio extends Client {
  private readonly bucket: string
  constructor({ port = 9000, useSSL = true, ...options }: MinioOptions) {
    super({ ...options, port: +port || 9000, useSSL)) })
    this.bucket = options.bucket
  }
  async download(name: string): Promise<DownloadSteam> {
    const [stat, stream] = await Promise.all([
      super.statObject(this.bucket, name),
      super.getObject(this.bucket, name)
    ])
    const downloadStream = new DownloadSteam(stat)
    stream.pipe(downloadStream)
    return downloadStream
  }

  upload(name: string, stream: Stream, meta?: { [k: string]: string }){
    return super.putObject(this.bucket, name, stream, meta)
  }
  async createURL(item: BucketItem): Promise<Item> {
    const stat = await this.statObject(this.bucket, item.name)
    return {
      ...item,
      url: await this.presignedGetObject(this.bucket,item.name,60*60*24*7,{
        Etag: stat.etag,
        'Content-Length': stat.size.toString(),
        'Content-Type': stat.metaData['content-type'],
        'Last-Modified': stat.lastModified.toISOString()
      })
    }
  }
  listAll(prefix?: string): Promise<Item[]> {
    const objects: BucketItem[] = []
    return new Promise<Item[]>((resolve, reject) =>
      super
        .listObjectsV2(this.bucket, prefix, true)
        .on('end', () => resolve(Promise.all(objects.map((obj) =>
this.createURL(obj)))))
        .on('data', (obj) => objects.push(obj))
        .on('error', reject)
    )
  }
  async boot(bucket: string = this.bucket): Promise<this> {
    if (!(await super.bucketExists(bucket))) await super.makeBucket(bucket)
    return this
  }
  static async getClient(connection: IConnection): Promise<Minio> {
    if (connection.options && connection.options.isLocal)
      await LocalStorageService.connect(connection.options)
    return new Minio({ bucket: 'test', ...connection.options }).boot()
  }
}
```

Лістинг 2. Модуль для запуску сторонніх програм

```
import * as os from 'os'
import { spawn, ChildProcess, SpawnOptions } from 'child_process'

export const RunScriptPrefix = {
  Linux: '.',
  Darwin: '.',
  Windows_NT: ''
} as { [key: string]: string }

export class Script {
  args: string[]
  private readonly command: string
  private instance: ChildProcess | null = null

  constructor(command: string, args: string[] = []) {
    this.args = args
    this.command = RunScriptPrefix[os.type()] + command
  }

  isActive(): boolean {
    return this.instance !== null && this.instance.connected &&
!this.instance.killed
  }

  start(options: SpawnOptions = {}): ChildProcess {
    this.instance = spawn(this.command, this.args, { cwd: '/', ...options })

    this.instance.stdout?.pipe(process.stdout)
    this.instance.stderr?.pipe(process.stderr)

    return this.instance
  }

  end(): boolean {
    return !this.isActive() || (this.instance as ChildProcess).kill()
  }
}
```

Лістинг 3. Модуль для роботи з локальною базою даних

```
import * as Nedb from 'nedb'

export class Cursor<T> implements PromiseLike<any[]> {
  cursor: Nedb.Cursor<T>

  constructor(cursor: Nedb.Cursor<T>) {
    this.cursor = cursor
  }

  skip(n: number): Cursor<T> {
    this.cursor.skip(n)
    return this
  }

  limit(n: number): Cursor<T> {
    this.cursor.limit(n)
    return this
  }

  sort(sort: any): Cursor<T> {
    this.cursor.sort(sort)
    return this
  }

  projection(projection: any): Cursor<T> {
    this.cursor.projection(projection)
    return this
  }

  then(resolve: (docs: T[]) => any, reject?: (err: Error) => any):
Promise<any> {
    return this.exec().then(resolve, reject)
  }

  catch(reject: (err: Error) => any): Promise<any> {
    return this.exec().catch(reject)
  }

  exec() {
    return new Promise<T[]>((resolve, reject) => {
      this.cursor.exec((err, ...args) => {
        if (err) reject(err)
        else resolve(...args)
      })
    })
  }
}
```

```
import * as Nedb from 'nedb'
import { Cursor } from './Cursor'

export class Datastore<T> {
  nedb: Nedb<T>
  constructor(opts?: string | Nedb.DataStoreOptions) {
    this.nedb = new Nedb<T>(opts)
  }

  findOne(filter: any, projection?: any): Promise<T | null> {
    return new Promise((resolve, reject) => {
      this.nedb.findOne(filter, projection, (err: any, ...args: any[]) => {
        if (err) reject(err)
        else resolve(...args)
      })
    })
  }

  find(filter: any, projection?: any): Cursor<T> {
    return new Cursor(this.nedb.find(filter, projection))
  }

  count(filter: any): Promise<number> {
    return new Promise<number>((resolve, reject) => {
      this.nedb.count(filter, (err: any, ...args: any[]) => {
        if (err) reject(err)
        else resolve(...args)
      })
    })
  }
  update(
    filter: any,
    update: any,
    options?: Nedb.UpdateOptions
  ): Promise<{ numAffected: number; upsert: boolean }>
  update(
    filter: any,
    update: any,
    options: Nedb.UpdateOptions
  ): Promise<{ numAffected: number; document: any; upsert: boolean }>

  update(
    filter: any,
    update: any,
    options: Nedb.UpdateOptions
  ): Promise<{ numAffected: number; documents: any[]; upsert: boolean }>

  update(
    filter: any,
    update: any,
    options?: { multi?: boolean; returnUpdatedDocs?: boolean; upsert?:
boolean }
  ): Promise<{
    numAffected: number
    document?: any
    documents?: any[]
    upsert: boolean
  }> {
    options = options || {}
    return new Promise((resolve, reject) => {
      this.nedb.update(
        filter,
        update,
        options as any,
```

```
        (err: any, numAffected: any, result: any, upsert: any) => {
          if (err) reject(err)
          else if (!options || !options.returnUpdatedDocs) {
            resolve({ numAffected, upsert })
          } else if (options.multi) {
            resolve({ numAffected, upsert, documents: any[] })
          } else {
            resolve({ numAffected, upsert, document: any })
          }
        }
      )
    })
  }
  remove(filter: any, options?: Nedb.RemoveOptions): Promise<number> {
    return new Promise((resolve, reject) => {
      this.nedb.remove(filter, options || {}, (err: any, n: number) => {
        if (err) reject(err)
        else resolve(n)
      })
    })
  }
  insert(newDoc: any): Promise<any>
  insert(newDocs: any[]): Promise<any[]>
  insert(insert: any | any[]): Promise<any | any[]> {
    return new Promise((resolve, reject) => {
      // here too is ts fault
      this.nedb.insert(insert as any, (err: any, res: any) => {
        if (err) reject(err)
        else resolve(res)
      })
    })
  }
  loadDatabase(): Promise<void> {
    return new Promise((resolve, reject) => {
      this.nedb.loadDatabase((err) => {
        if (err) reject(err)
        else resolve()
      })
    })
  }
  ensureIndex(options: any): Promise<void> {
    return new Promise((resolve, reject) => {
      this.nedb.ensureIndex(options, (err: any) => {
        if (err) reject(err)
        else resolve()
      })
    })
  }
  removeIndex(fieldName: string): Promise<void> {
    return new Promise((resolve, reject) => {
      this.nedb.removeIndex(fieldName, (err: any) => {
        if (err) reject(err)
        else resolve()
      })
    })
  }
}
```

Лістинг 4. Клас для роботи з документами у локальній базі даних

```
import * as NeDB from 'nedb'
import { Store } from './Store'

export interface IDocument {
  _id?: string
  createdAt?: Date
  updatedAt?: Date
}

export interface UpdateOptions extends NeDB.UpdateOptions {
  new: boolean
}

export class Document<T extends IDocument = IDocument> extends Store<T> {
  protected constructor(dbPath: string) {
    super(dbPath)
  }

  findById(_id: string): Promise<T | null> {
    return super.findOne({ _id })
  }

  create(data: T): Promise<T | T[]> {
    return super.insert(data)
  }

  async findOneAndUpdate(
    filter: Partial<T>,
    data: Partial<T>,
    options?: UpdateOptions
  ): Promise<T | null> {
    const current = await super.findOne(filter)

    if (!current && !options?.upsert) return null

    const result = await super.update(filter, data, options)

    if (!result.numAffected) return null

    return options?.new ? await super.findOne(filter) : current
  }

  async findByIdAndUpdate(_id: string, data: Partial<T>, options?:
UpdateOptions) {
    const current = await super.findOne({ _id })

    if (!current && !options?.upsert) return null

    const result = await super.update({ _id }, data, options)

    if (!result.numAffected) return null

    return options?.new ? await super.findOne({ _id }) : current
  }
}
```

Лістинг 5. Модуль для обробки завантаження файлів

```
import * as Busboy from 'busboy'
import { Request } from 'express'

import { IConnection, Minio } from '../providers'

export class UploadControllerStatic {
  async processUpload(req: Request, connection: IConnection) {
    const client = await Minio.getClient(connection)

    return new Promise((resolve, reject) => {
      const files: string[] = []

      const busboy = new Busboy({
        headers: req.headers
      })

      busboy
        .on('file', async (field, file, name, encoding, mimeType) => {
          try {
            await client.upload(name, file.on('end', () =>
files.push(name)).on('error', reject), {
              mimeType
            })
          } catch (e) {
            console.error('UPLOADING ERROR', e)
          }
        })
        .on('error', reject)
        .on('finish', () => resolve(files))

      req.pipe(busboy)
    })
  }
}

export const UploadController = new UploadControllerStatic()
```

Лістинг 6. Модуль для запуску локального сховища даних

```
import * as os from 'os'
import * as path from 'path'

import { Script } from '../helpers'
import { StorageOptions } from '../interfaces'
import { Connection } from '../providers/Nedb'

export const MinioBinary = {
  Darwin: 'minio',
  Linux: 'minio.sh',
  Windows_NT: 'minio.exe'
} as { [key: string]: string }

export const MinioBinaryPath = path.resolve(process.cwd(),'bin','minio',
MinioBinary[os.type()])
export class LocalStorageServiceStatics {
  private readonly script: Script
  constructor(private readonly appDirPath: string) {
    this.script = new Script(MinioBinaryPath, [
      'server',
      this.dataDirPath,
      '--config-dir',
      this.configDirPath
    ])
  }
  get dataDirPath(): string {
    return path.resolve(this.appDirPath, 'minio')
  }
  get configDirPath(): string {
    return path.resolve(this.appDirPath, 'config')
  }
  connect(options?: Partial<StorageOptions>): this {
    const env: any = {
      MINIO_ACCESS_KEY: (options && options.accessKey) ||
process.env.MINIO_ACCESS_KEY,
      MINIO_SECRET_KEY: (options && options.secretKey) ||
process.env.MINIO_SECRET_KEY
    }
    if (options) {
      if (options.port) {
        this.script.args.push('--address', `:${options.port}`)
      }
      if (options.region) env.MINIO_REGION_NAME = options.region
      if (options.domain) env.MINIO_DOMAIN = options.domain
      if (!options.browser) env.MINIO_BROWSER = 'off'
    }
    this.script.start({ env })
    return this
  }

  static async getLocalConnections() {
    return (await Connection.find({ 'options.isLocal': true })).map((c) =>
c.options)
  }
}

export const LocalStorageService = new LocalStorageServiceStatics(
  path.resolve(os.homedir(), '.config', 'shares3')
)
```

Лістинг 7. Модуль для запуску балансувальника запитів

```
import * as os from 'os'
import * as path from 'path'
import * as YAML from 'yaml'

import { DistributorServiceOptions } from '../interfaces'
import { Script, generateLocalConfigFile } from '../helpers'

export const RadioBinary = {
  Linux: 'radio.sh'
} as { [key: string]: string }

export const RadioBinaryPath = path.resolve(process.cwd(), 'bin', 'radio',
RadioBinary[os.type()])

export class DistributorServiceStatics {
  private readonly script: Script

  constructor(options: DistributorServiceOptions) {
    const configFile = this.prepareInstructions(options)
    this.script = new Script(RadioBinaryPath, ['server', '-c', configFile])
  }

  prepareInstructions(options: DistributorServiceOptions): string {
    const content = YAML.stringify(options)
    return generateLocalConfigFile('config.yml', content)
  }

  connect() {
    return this.script.start()
  }

  disconnect() {
    return this.script.end()
  }

  static prepareRemotes(
    connectionOptions: any = [],
    { accessKey, secretKey }: { [k: string]: string } = {}
  ) {
    return new DistributorServiceStatics({
      buckets: {
        common: {
          access_key: accessKey,
          secret_key: secretKey,
          protection: {
            scheme: 'mirror'
          },
          remote: connectionOptions.map((options: any) => ({
            bucket: options.bucket,
            endpoint: options.endPoint,
            access_key: options.accessKey,
            secret_key: options.secretKey
          }))
        }
      }
    })
  }
}
```

Лістинг 8. Модуль для роботи сховища з мережею

```
import { isIP } from 'net'
import * as url from 'url'
import * as ngrok from 'ngrok'

export const STATIC_PUBLIC_IP = process.env.STATIC_PUBLIC_IP

export class GatewayServiceStatics {
  private gateway?: string
  private isEnabled = false

  constructor(private readonly publicIP = STATIC_PUBLIC_IP) {}

  async connect(port: number): Promise<string> {
    if (this.publicIP && isIP(this.publicIP)) {
      this.gateway = url.format({
        port,
        protocol: 'http',
        hostname: this.publicIP
      })
    }

    if (!this.gateway) {
      this.gateway = await ngrok.connect({
        addr: port,
        proto: 'http'
      })
      this.isEnabled = true
    }

    return this.gateway
  }

  async disconnect(): Promise<void> {
    if (this.isEnabled && this.gateway) {
      await ngrok.disconnect(this.gateway)
      this.isEnabled = false
    }
    this.gateway = undefined
  }
}
```

Лістинг 9. Модель для роботи з сутністю «User»

```
import * as bcrypt from 'bcrypt'

import * as config from '../config'
import { Document, IDocument } from '../lib'
import { createToken } from '../../../helpers'
import { AuthorizationError } from '../../../config'

export interface IUser extends IDocument {
  login: string
  password: string
}

export class User extends Document<IUser> {
  constructor() {
    super(config.USERS_DB_PATH)
  }

  async authorize(login: string, password: string): Promise<any> {
    let user: any = await super.findOne({ login })

    console.log({ login, password })

    if (user) {
      if (!(await bcrypt.compare(password, user.password)))
        throw new AuthorizationError('Password invalid!')
    } else {
      user = await super.insert({
        login,
        password: await bcrypt.hash(password, 10)
      })
    }

    return {
      user,
      token: createToken({ id: user._id })
    }
  }
}
```

Лістинг 10. Модель для роботи з сутністю «Connection»

```
import { ClientOptions } from 'minio'

import { IUser } from './User'
import * as config from '../config'
import { BaseError } from '../../../config'
import { Document, IDocument } from '../lib'

export enum ConnectionType {
  DEFAULT = 'default',
  ACTIVE = 'active',
  INACTIVE = 'inactive'
}

export interface ConnectionOptions extends ClientOptions {
  isLocal?: boolean
}

export interface IConnection extends IDocument {
  domain?: string
  browser?: boolean
  type: ConnectionType
  userId: string
  name: string
  options: ConnectionOptions
}

export class Connection extends Document<IConnection> {
  constructor() {
    super(config.CONNECTIONS_DB_PATH)
  }

  async updateDefault(_id: string, user: IUser): Promise<IConnection> {
    const userId = user._id
    const connection = await super.findOne({ _id, userId })
    if (!connection) throw new BaseError('Connection not found!', 404)
    if (connection.type === ConnectionType.DEFAULT) return connection

    await super.update(
      {
        type: ConnectionType.DEFAULT,
        userId
      },
      { type: ConnectionType.ACTIVE }
    )

    const newDefault = await super.findByIdAndUpdate(
      _id,
      { type: ConnectionType.DEFAULT },
      { new: true }
    )

    return newDefault as IConnection
  }
}
```

Лістинг 11. Клас для покращення роботи при потоковому завантаженні даних

```
import { PassThrough } from 'stream'
import { BucketItemStat } from 'minio'

export interface DownloadSteamMetadata {
  Etag?: string
  'Content-Type'?: string
  'Last-Modified'?: string
  'Content-Length'?: string
}

export class DownloadSteam extends PassThrough {
  private readonly metaData: DownloadSteamMetadata

  constructor(stat: BucketItemStat) {
    super()

    this.metaData = {
      Etag: stat.etag,
      'Content-Length': stat.size.toString(),
      'Content-Type': stat.metaData['content-type'],
      'Last-Modified': stat.lastModified.toISOString()
    }
  }

  getMetaData(): DownloadSteamMetadata {
    return this.metaData
  }
}
```

Лістинг 12. Команди для автоматичного налаштування платформи Go, встановлення балансувальника запитів

```
#!/usr/bin/env bash

if ! [$(command -v go)]; then
 wget -O golang.tar.gz https://dl.google.com/go/go1.14.4.linux-amd64.tar.gz
 tar -C /usr/local -xzf golang.tar.gz
 export PATH=$PATH:/usr/local/go/bin
fi

GO111MODULE=on go get github.com/minio/radio
mv go/bin/radio ../bin/radio
chmod +x radio
```

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

Децентралізована система для зберігання та поширення даних на основі технології S3

Виконав: Кириченко Владислав Андрійович, КП-62

Керівник: старший викладач каф. ПЗКС ФПМ Ю.В. Бухтіяров

Київ – 2020

ПОСТАНОВКА ЗАДАЧІ

Мета проекту: Основна мета дипломної роботи - розробка програмного забезпечення для зручного та надійного зберігання даних, використовуючи для цього лише власний пристрій або пристрої що використовують це ж ПЗ

Існуючі рішення

napster™

BitTorrent®

I2P

Проблеми, що вирішує проєкт

- Безпечне та ізольоване від інших програм зберігання даних на власному пристрої
- Можливість надавання безпечного доступу для обраних файлів на власному пристрої в обмежений публічний доступ
- Можливість децентралізовано зберігати вміст одночасно на декількох пристроях для підвищення надійності

АКТУАЛЬНІСТЬ

Додаток “ShareS3” для зберігання та надавання доступу до окремих файлі – нове рішення, що об'єднує у собі можливості peer-to-peer з'єднання та універсальне і просте для інтеграції API сховищ класу S3, що дозволяє використовувати вбудовані можливості сховищ задля розподіленого зберігання та роботи з даними

Постановка задачі

Мета: забезпечити можливість завантаження будь-яких даних до власного сховища, перегляду їх та вивантаження на необхідний пристрій, використовуючи для цього лише пристрій, що власне зберігає дані, та пристрій з якого вони переглядаються

Завдання:

1. Проаналізувати існуючі програмні рішення
2. Розробити програмний застосунок відповідно до вимог
3. Протестувати роботу веб-додатку

Вимоги до продукту

- Забезпечення зручності, якості та швидкого розуміння роботи з програмою
- Можливість перегляду файлів на пристрої використовуючи безпосереднє з'єднання між пристроями
- Можливість завантажувати будь-які необхідні файли
- Можливість запускати розподілене між декількома пристроями сховище

Вимоги до інтерфейсу

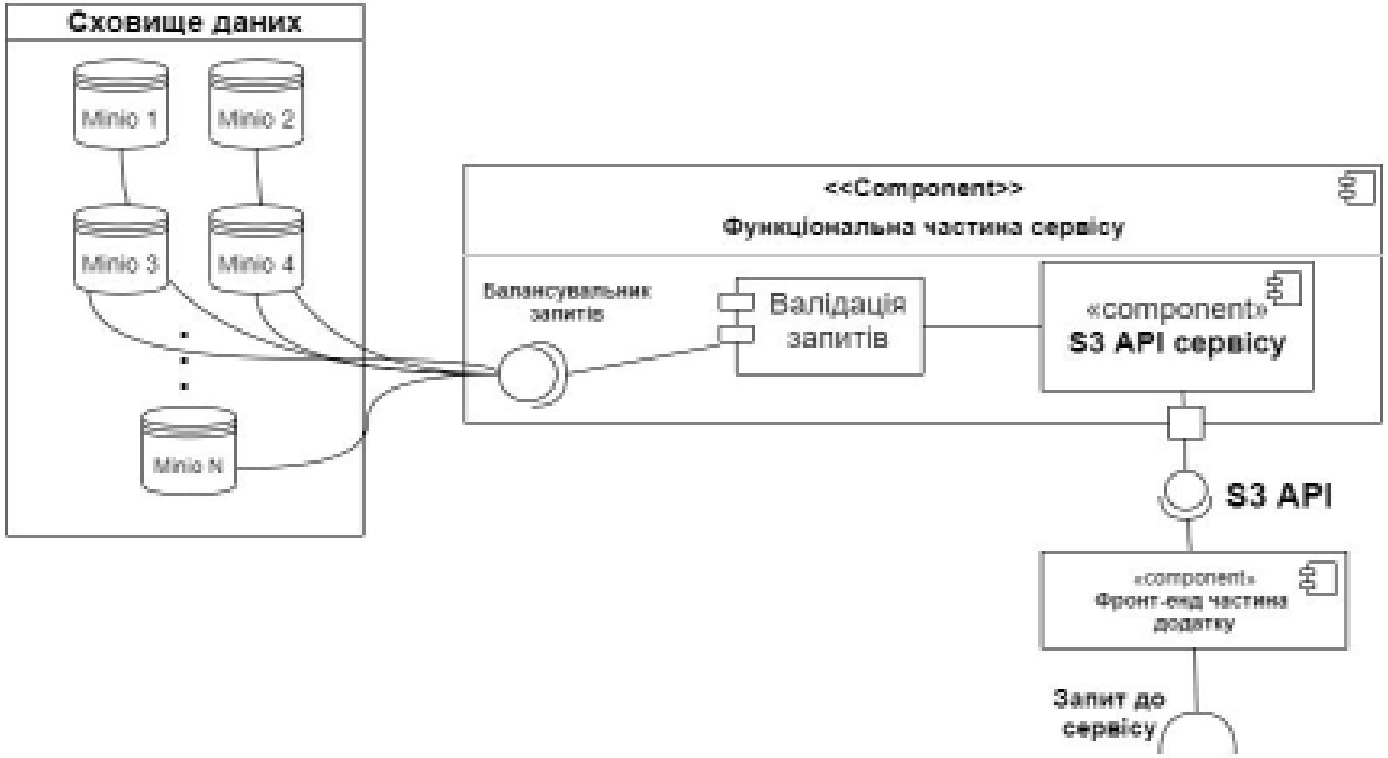
- Інтерфейс системи повинен мати інтуїтивно зрозумілі елементи управління
- Можливість адміністрування доступу до вмісту
- Реалізувати доступ до файлів у схожому до ftp вигляді

Вибір технологій

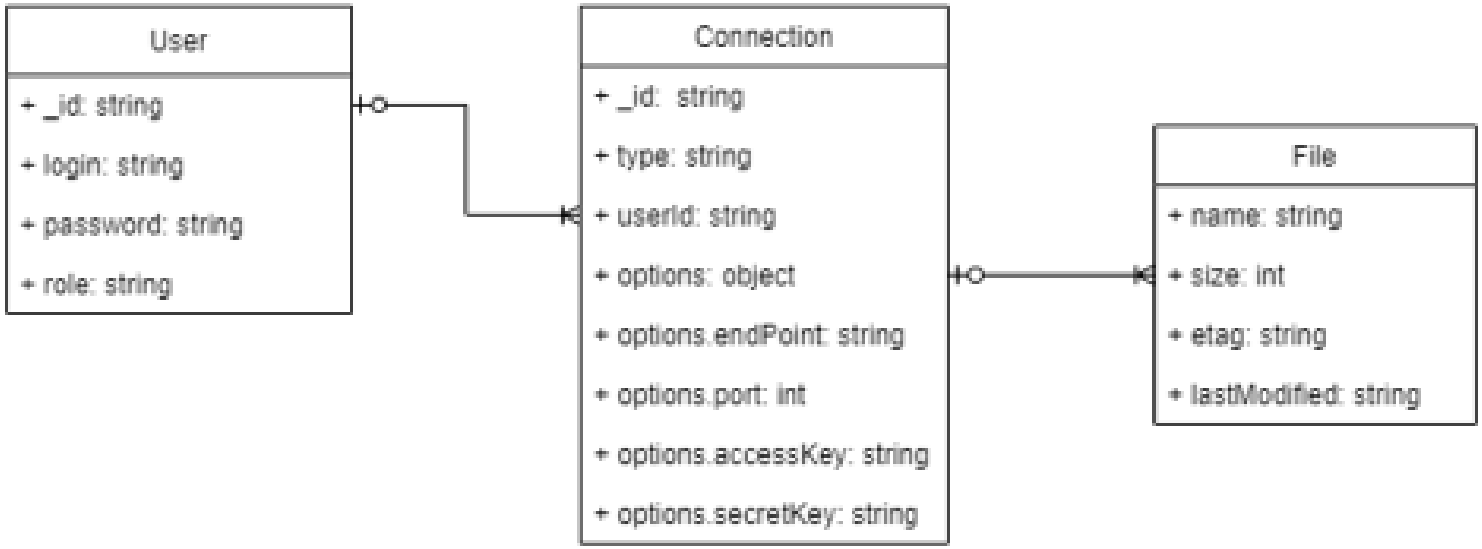
- Мова програмування: Node.JS, TypeScript
- Локальна база даних: Nedb
- Засіб для реалізації сховища: Minio
- Фреймворк для мережевої взаємодії: Fastify
- Засіб для потокової обробки даних: Busboy
- Фреймворк для інтерфейсу взаємодії: Vue.JS

Дерево проблем

Архітектура децентралізованого сховища даних



Діаграма структури СКБД

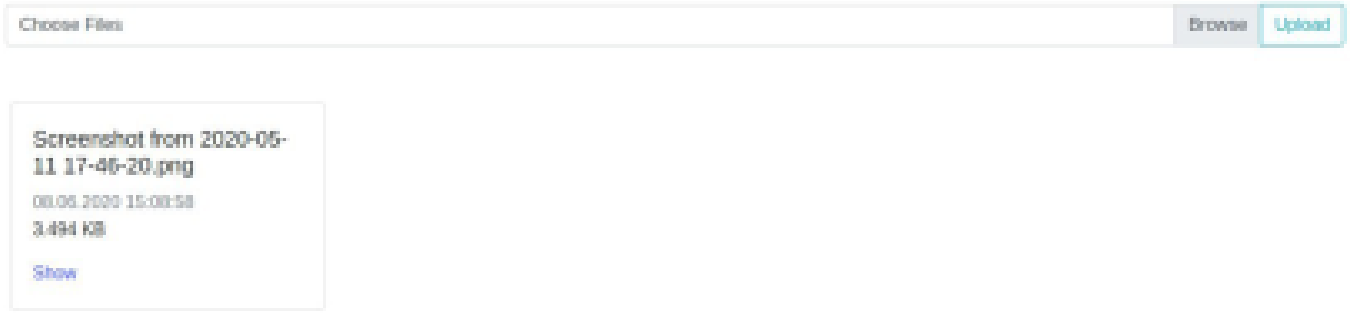


Приклад роботи

Форма авторизації користувача

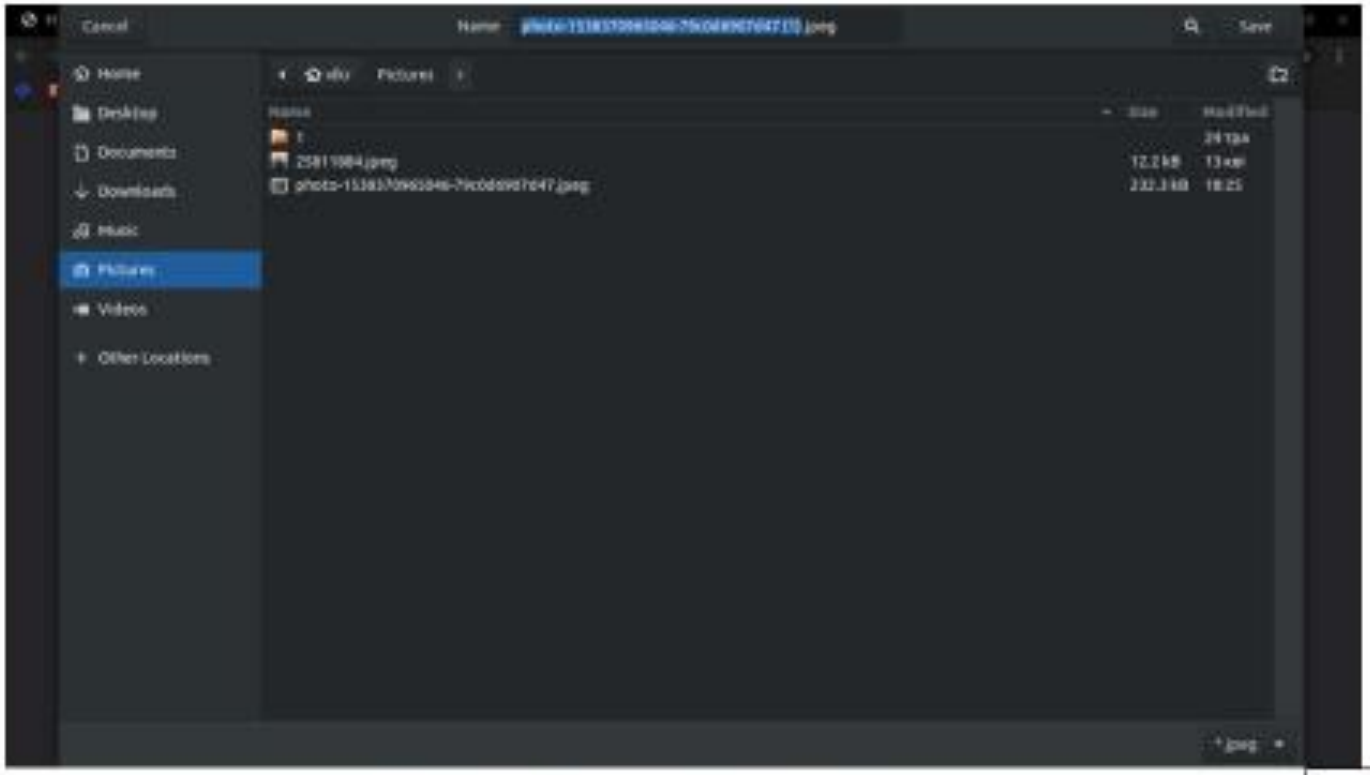
Вибір підключення

Приклад роботи



Перегляд доступних файлів

Приклад роботи



Форма для завантаження

Порівняння з аналогами

Назва	Шифрування даних	Розподілене зберігання	Завантаження даних	Наявність центрального серверу
Napster	-	-	+	+
BitTorrent	-	+	+	+/-
Kademlia	+	-	+	-
Розроблене ПЗ	+	+	+	-

Результат перевірки на унікальність

ВИСНОВКИ

1. Проаналізовано існуючі програмні рішення
2. Розроблено архітектуру програми
3. Реалізовано інтерфейс для взаємодії з програмою
4. Реалізовано вимоги до програми
5. Програма у вигляді настільного додатку
6. Протестовано роботу програми

Дякую за увагу!